GenoGAM: Genome-wide generalized additive models

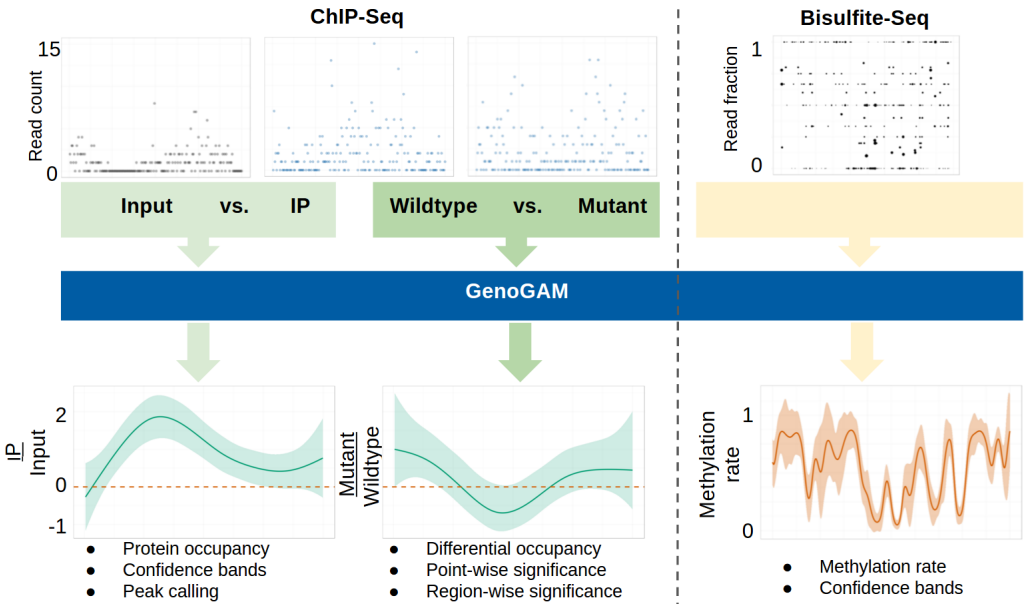
Georg Stricker[1], Julien Gagneur[1]

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

May 3, 2017

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 workflow.



GenoGAM version: 1.2.1

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

Stricker and Engelhardt, *et al.* **GenoGAM: Genome-wide generalized additive models for ChIP-seq analysis** *Bioinformatics*

Contents

1 TL;DR analysis

This is the brief version of the usual workflow of *GenoGAM*. It involves:

- Reading in data through `GenoGAMDataSet` to get a `GenoGAMDataSet` object
- Computing size factors with `computeSizeFactors`
- Compute model with `genogam` to get the result `GenoGAM` object

```
library(fastGenoGAM)

## specify folder and experiment design path
folder <- system.file("extdata/Set1", package='GenoGAM')
expDesign <- file.path(folder, "experimentDesign.txt")

## specify chunk and overhang size, bpk being the knot spacing
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 [2017-05-03 15:57:48] Creating GenoGAMDataSet
## INFO [2017-05-03 15:57:49] Reading in data
## INFO [2017-05-03 15:57:49] Reading in wt_1
## INFO [2017-05-03 15:57:50] Reading in wt_2
## INFO [2017-05-03 15:57:51] Reading in mutant_1
## INFO [2017-05-03 15:57:51] Reading in mutant_2
## INFO [2017-05-03 15:57:51] Finished reading in data
## INFO [2017-05-03 15:57:51] GenoGAMDataSet created
```

```
ggd
```

```
## class: GenoGAMDataSet
## dimension: 784333 4
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## design variables(1): genotype
## tiles size: 1.6kbp
## number of tiles: 785
## chromosomes: chrXIV
## size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##        0        0        0        0
## formula:
## ~ s(x) + s(x, by = genotype)
```

```
## compute size factors
ggd <- computeSizeFactors(ggd)
```

```
## INFO [2017-05-03 15:57:51] Computing size factors
## INFO [2017-05-03 15:57:51] DONE
```

```
ggd
```

```
## class: GenoGAMDataSet
## dimension: 784333 4
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## design variables(1): genotype
## tiles size: 1.6kbp
## number of tiles: 785
## chromosomes: chrXIV
## size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##  -0.0702   0.1393  -0.0161   0.2175
## formula:
## ~ s(x) + s(x, by = genotype)
```

```
## restrict to the actual small data region (usually not needed, just for vignette purposes)
ggd <- subset(ggd, seqnames == "chrXIV" & pos >= 305000 & pos <= 308000)
```

```
ggd
```

```
## class: GenoGAMDataSet
## dimension: 3001 4
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## design variables(1): genotype
## tiles size: 1.6kbp
## number of tiles: 3
## chromosomes: chrXIV
## size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##  -0.0702   0.1393  -0.0161   0.2175
## formula:
## ~ s(x) + s(x, by = genotype)
```

```
## compute model without parameter estimation to save time in vignette
result <- genogam(ggd, lambda = 4601, theta = 4.51)
```

```
## INFO [2017-05-03 15:57:52] Initializing the model
## INFO [2017-05-03 15:57:53] Done
## INFO [2017-05-03 15:57:53] Fitting model
## INFO [2017-05-03 15:58:12] Done
## INFO [2017-05-03 15:58:12] Assembling fits and building GenoGAM object
## INFO [2017-05-03 15:58:12] Done
```

```
result
```

```
## Family:  Negative Binomial
```

```
## Formula:  ~ s(x) + s(x, by = genotype)
## Class: GenoGAM
## Dimension: 3001 4
## Samples(4): wt_1 wt_2 mutant_1 mutant_2
## Design variables(1): genotype
## Smooth functions(2): s(x) s(x):genotype
## Chromosomes: chrXIV
##
## Size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##  -0.0702   0.1393  -0.0161   0.2175
##
## Cross Validation:  Not performed
##
## Spline Parameters:
##   Knot spacing:  20
##   B-spline order:  2
##   Penalization order:  2
##
## Tile settings:
##   Chunk size: 1000
##   Tile size: 1600
##   Overhang size: 300
##   Number of tiles: 3
##   Evaluated genome ranges:
## GRanges object with 1 range and 0 metadata columns:
##       seqnames           ranges strand
##          <Rle>        <IRanges>  <Rle>
##   [1]   chrXIV [305000, 308000]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

2 Standard ChIP-Seq analysis

2.1 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:

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

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.

2.2 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 collector, 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 multicore machines by default the number of available cores - 2 are registered In order to see which parallel backend is currently registered, use the `registered` function. The first entry is the currently registered backend. We can also directly just take the first entry by subsetting:

```
library(fastGenoGAM)

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

```
## $MulticoreParam
## class: MulticoreParam
##   bpisup: FALSE; bpnworkers: 6; 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 enable a progressbar. Check *BiocParallel* for other possible backends and more options for `SnowParam`. Implementing a progressbar in parallel is a little bit tricky and requires the master process to constantly monitor the state of the worker processes. Since this adds to overhead, monitoring does not really happen in real time. So don't expect the progressbar to be precise.

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

If we check the current registered backend, we see that the number of workers and the backend 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
```

2.3 Building a GenoGAMDataSet 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='fastGenoGAM')

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.

Four more parameters are required to build a `GenoGAMDataSet`:

- The chunk size in basepairs, i.e. the size of the single region on which the model will be computed. See subsection below for a discussion on the right choice of this parameter.
- The overhang size, i.e. the overlap in basepairs on one side. It will be extended automatically to both sides. See subsection below for a discussion on the right choice of this parameter.

- The design, i.e. the model formula.
- The directory of the BAM files. This is not mandatory, but by default, *GenoGAM* will assume the data in the working directory path, which is usually not the case

2.3.1 Reading in data from files

We will now count sequencing fragment centers per genomic position and sample and store these counts into a *GenoGAMDataSet*. By default *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()`). We choose a knot spacing of 20bp, a chunk size of 1kb and and overhang of 15 knots, i.e 300bp. The *experimentDesign* argument doesn't need to be a `data.frame`, but can be just the path to the experiment design file. Although the knot spacing won't play a role at this stage, it is already a good idea to define the overhang size in terms of it (see section 'How to choose chunk and overhang size' for more).

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

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

```
## INFO [2017-05-03 15:58:12] Creating GenoGAMDataSet
## INFO [2017-05-03 15:58:13] Reading in data
## INFO [2017-05-03 15:58:13] Reading in wt_1
## INFO [2017-05-03 15:58:13] Reading in wt_2
## INFO [2017-05-03 15:58:13] Reading in mutant_1
## INFO [2017-05-03 15:58:14] Reading in mutant_2
## INFO [2017-05-03 15:58:14] Finished reading in data
## INFO [2017-05-03 15:58:14] GenoGAMDataSet created
```

```
ggd
```

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

```
## ~ s(x) + s(x, by = genotype)
```

If not otherwise specified the BAM files are read with respect to the BAM file header. We can see that only one chromosome has been read in, chrXIV. Thus the dimension of the `GenoGAMDataSet` object states the length of the chromosome (for rows) and the number of columns, i.e. samples. The tiles are of size 1.6kb, which is the sum of the specified chunk size and twice the overhang. However the actual data covered by the BAM file is much smaller, so we reduce the `GenoGAMDataSet` object. The positions are obviously known by design. See the *Settings* subsection below for methods to specify regions prior to reading.

2.3.2 Subsetting

Subsetting works via the `subset` command or by `GRanges`. The latter works through the `subsetByOverlaps` functions. Thus it can be slower, as it has to determine the overlaps first.

```
## Subsetting by subset command requires to know the column names
rowRanges(ggd)
```

```
## GPos object with 784333 positions and 0 metadata columns:
##            seqnames       pos strand
##               <Rle> <integer>  <Rle>
##        [1]   chrXIV         1      *
##        [2]   chrXIV         2      *
##        [3]   chrXIV         3      *
##        [4]   chrXIV         4      *
##        [5]   chrXIV         5      *
##        ...      ...       ...    ...
##   [784329]   chrXIV    784329      *
##   [784330]   chrXIV    784330      *
##   [784331]   chrXIV    784331      *
##   [784332]   chrXIV    784332      *
##   [784333]   chrXIV    784333      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## Here we know the positions by design of this example
subggd <- subset(ggd, seqnames == "chrXIV" & pos >= 305001 & pos <= 308000)
subggd
```

```
## class: GenoGAMDataSet
## dimension: 3000 4
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## design variables(1): genotype
## tiles size: 1.6kbp
## number of tiles: 3
## chromosomes: chrXIV
## size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##        0        0        0        0
## formula:
```

```
## ~ s(x) + s(x, by = genotype)
```

```
## Or subset by GRanges
gr <- GenomicRanges::GRanges("chrXIV", IRanges(305001, 308000))
gr
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames           ranges strand
##          <Rle>        <IRanges>  <Rle>
##   [1]   chrXIV [305001, 308000]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
subggd <- ggd[gr] ## same as subsetByOverlaps(ggd)
subggd
```

```
## class: GenoGAMDataSet
## dimension: 3000 4
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## design variables(1): genotype
## tiles size: 1.6kbp
## number of tiles: 3
## chromosomes: chrXIV
## size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##        0        0        0        0
## formula:
## ~ s(x) + s(x, by = genotype)
```

2.3.3 Reading in data from SummarizedExperiment

Sometimes data comes in the form of a *RangedSummarizedExperiment* or users like to read in the data themselves instead of using the package default. In this case the `GenoGAMDataSet` object can be built the same way.

```
## load the SummarizedExperiment object, a new variable "se" appears in the environment
load(file.path(folder, "se.rda"))
se
```

```
## class: RangedSummarizedExperiment
## dim: 784333 4
## metadata(0):
## assays(1): ''
## rownames: NULL
## rowData names(0):
## colnames(4): wt_1 wt_2 mutant_1 mutant_2
## colData names(1): genotype
```

```
## build the GenoGAMDataSet from SummarizedExperiment object.
## Other parameters remain the same
```

```
se_ggd <- GenoGAMDataSet(
  experimentDesign = se, chunkSize = chunkSize,
  overhangSize = overhangSize,
  design = ~ s(x) + s(x, by = genotype)
)
```

```
## INFO [2017-05-03 15:58:15] Creating GenoGAMDataSet
## INFO [2017-05-03 15:58:16] GenoGAMDataSet created
```

```
se_ggd
```

```
## class: GenoGAMDataSet
## dimension: 784333 4
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## design variables(1): genotype
## tiles size: 1.6kbp
## number of tiles: 785
## chromosomes: chrXIV
## size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##        0        0        0        0
## formula:
## ~ s(x) + s(x, by = genotype)
```

2.3.4 The GenoGAMDataSet object

A *GenoGAMDataSet* stores the 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 sufficient 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 `GenoGAMDataSet` object contains three major objects (the data, the positions and the index) and a couple of useful slots (design, size factors, formula, etc.). As *GenoGAMDataSet* is build on top of *RangedSummarizedExperiment* most methods work the same way:

```
## The data
assay(subggd)
```

```
## DataFrame with 3000 rows and 4 columns
##       wt_1  wt_2 mutant_1 mutant_2
##      <Rle> <Rle>    <Rle>    <Rle>
## 1        0     0        0        0
```

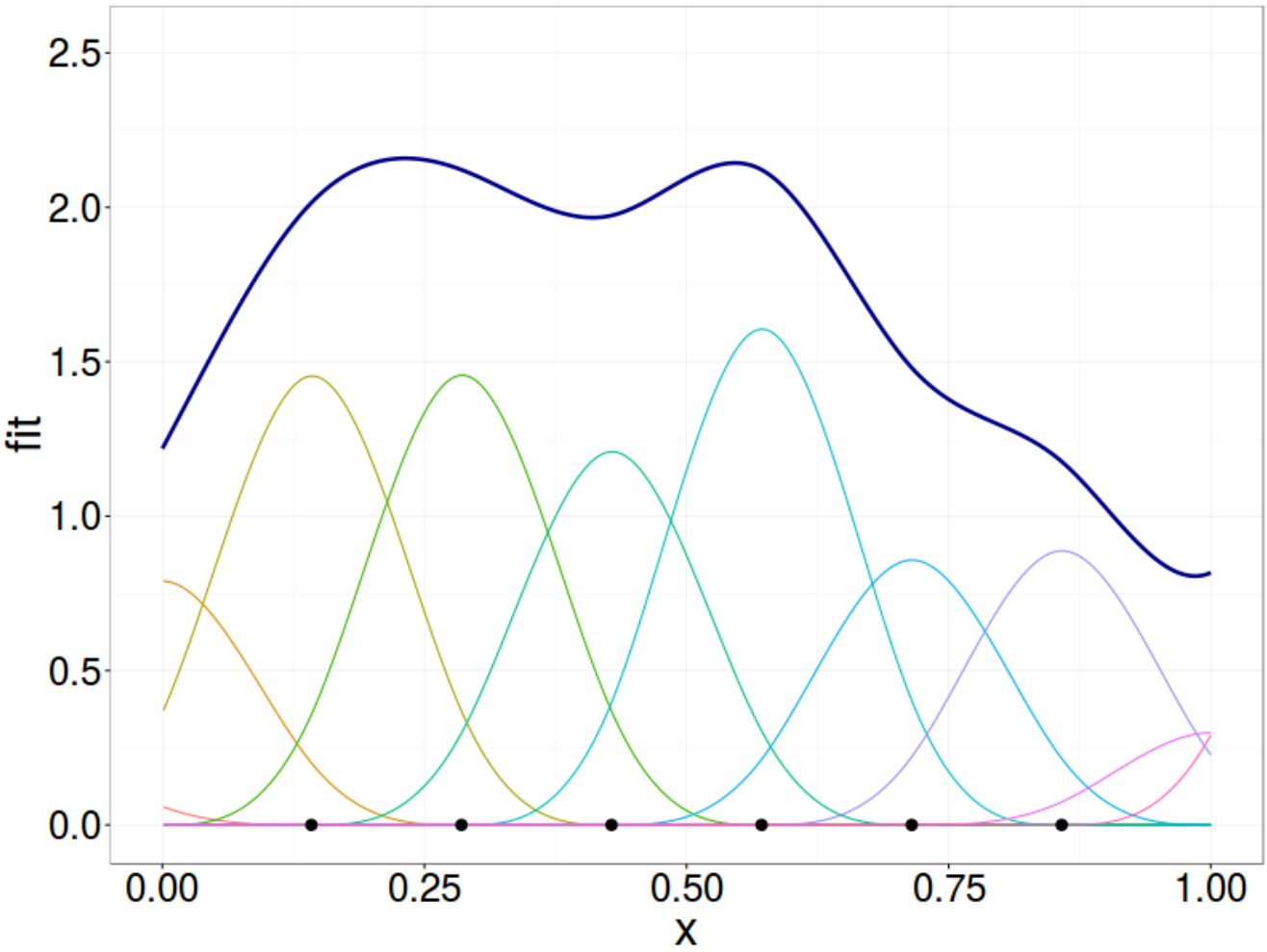


Figure 1: An example spline. Displayed are ten cubic B-spline basis functions (of some only tails are seen at the borders) which multiplied by their respective coefficients and summed over at every position make up the complete spline (dark blue). The knots are depicted as dark-grey dots at the bottom-center of each basis function.

```
## 2        0     1        0        0
## 3        0     0        0        0
## 4        0     0        0        0
## 5        0     1        0        0
## ...    ...   ...      ...      ...
## 2996     0     0        0        0
## 2997     0     0        0        0
## 2998     1     0        0        0
## 2999     0     1        0        0
## 3000     0     0        0        0


## The positions
rowRanges(subggd)
```

```
## GPos object with 3000 positions and 0 metadata columns:
##          seqnames       pos strand
##             <Rle> <integer>  <Rle>
##      [1]   chrXIV    305001      *
##      [2]   chrXIV    305002      *
##      [3]   chrXIV    305003      *
##      [4]   chrXIV    305004      *
##      [5]   chrXIV    305005      *
##      ...      ...       ...    ...
##   [2996]   chrXIV    307996      *
##   [2997]   chrXIV    307997      *
##   [2998]   chrXIV    307998      *
##   [2999]   chrXIV    307999      *
##   [3000]   chrXIV    308000      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## The index
getIndex(subggd)
```

```
## GRanges object with 3 ranges and 1 metadata column:
##       seqnames           ranges strand |        id
##          <Rle>        <IRanges>  <Rle> | <integer>
##   [1]   chrXIV [305001, 306600]      * |         1
##   [2]   chrXIV [305701, 307300]      * |         2
##   [3]   chrXIV [306401, 308000]      * |         3
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## experiment design
colData(subggd)
```

```
## DataFrame with 4 rows and 1 column
##           genotype
##          <integer>
## wt_1             0
## wt_2             0
## mutant_1         1
## mutant_2         1
```

```
## formula
design(subggd)
```

```
## ~s(x) + s(x, by = genotype)
```

```
## size factors, initially zero
sizeFactors(subggd)
```

```
##     wt_1     wt_2 mutant_1 mutant_2
##        0        0        0        0
```

```
## some important settings
```

```
tileSettings(subggd)
```

```
## $chromosomes
## GRanges object with 1 range and 0 metadata columns:
##       seqnames           ranges strand
##          <Rle>        <IRanges>  <Rle>
##   [1]   chrXIV [305001, 308000]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
##
## $chunkSize
## [1] 1000
##
## $overhangSize
## [1] 300
##
## $tileSize
## [1] 1600
##
## $numTiles
## [1] 3
##
## $check
## [1] TRUE
```

The last command is a list of settings, where each element also has an individual method to be accessed (see ?GenoGAMDataSet). The element *check* tells you if the consistency of the individual objects against each other is checked, e.g. if the chromosomes in the positions object actually comply with the chromosomes in the index object. If this field is FALSE, then some condition got violated and it is not necessarily safe to compute the model anymore. A more important use of the single elements of the tile settings is the possibility to manipulate them. Note that those settings can never be fully enforced on the data as an exact tiling requires the chunk size to be a multiple of the length of one chromosomes. However chromosomes have different lengths. Thus there will be always differences, especially in the first and last two chunks of a chromosome. However the model does only require equal tile size, which is always guaranteed. On top the chunk size is guaranteed to hold on average.

```
## the original index
getIndex(subggd)
```

```
## GRanges object with 3 ranges and 1 metadata column:
##       seqnames           ranges strand |        id
##          <Rle>        <IRanges>  <Rle> | <integer>
##   [1]   chrXIV [305001, 306600]      * |         1
##   [2]   chrXIV [305701, 307300]      * |         2
##   [3]   chrXIV [306401, 308000]      * |         3
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## change chunk size, new tile size is now 2600
getChunkSize(subggd) <- 2000
```

```
getIndex(subggd)
```

```
## GRanges object with 2 ranges and 1 metadata column:
##       seqnames           ranges strand |        id
##          <Rle>        <IRanges>  <Rle> | <integer>
##   [1]   chrXIV [305001, 307600]      * |         1
##   [2]   chrXIV [305401, 308000]      * |         2
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## change tile size, new chunk size is now 1400
getTileSize(subggd) <- 2000
getIndex(subggd)
```

```
## GRanges object with 2 ranges and 1 metadata column:
##       seqnames           ranges strand |        id
##          <Rle>        <IRanges>  <Rle> | <integer>
##   [1]   chrXIV [305001, 307000]      * |         1
##   [2]   chrXIV [306001, 308000]      * |         2
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## change overhang size, chunk size is now equal to tile size
getOverhangSize(subggd) <- 0
getIndex(subggd)
```

```
## GRanges object with 3 ranges and 1 metadata column:
##       seqnames           ranges strand |        id
##          <Rle>        <IRanges>  <Rle> | <integer>
##   [1]   chrXIV [305001, 306400]      * |         1
##   [2]   chrXIV [306401, 307800]      * |         2
##   [3]   chrXIV [306601, 308000]      * |         3
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## Note how the overhang size can not be zero between the last two tiles,
## as the chunk size is set to 1400 within a 3kb window. This will
## have an impact in the next call.
getTileNumber(subggd) <- 1
## A warnings is issued as the settings indicate a longer total size of the genome.
getIndex(subggd)
```

```
## GRanges object with 1 range and 1 metadata column:
##       seqnames           ranges strand |        id
##          <Rle>        <IRanges>  <Rle> | <integer>
##   [1]   chrXIV [305001, 308000]      * |         1
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

Additionally a number of useful metrics is defined on *GenoGAMDataSet* on a a tile basis, e.g. the sum or the mean function. This also makes it possible to manually filter the regions.

```
## return tile number to three
getTileNumber(subggd) <- 3

## the sum
sum(subggd)
```

```
##   wt_1 wt_2 mutant_1 mutant_2
## 1   65   65       85      142
## 2  447  583      242      603
## 3  276  311      192      297
```

```
## or the mean
mean(subggd)
```

```
##    wt_1  wt_2 mutant_1 mutant_2
## 1 0.065 0.065    0.085    0.142
## 2 0.447 0.583    0.242    0.603
## 3 0.276 0.311    0.192    0.297
```

```
## A simple filter can be done based on this functionality. With the threshold
## being 0.05 reads per bp it results in the same region plus the overhang on
## both sides.
totalMeans <- mean(ggd)
idx <- which(rowMeans(totalMeans) > 0.05)
new_subggd <- subsetByOverlaps(ggd, getIndex(ggd)[idx])
getIndex(new_subggd)
```

```
## GRanges object with 4 ranges and 1 metadata column:
##       seqnames           ranges strand |        id
##          <Rle>        <IRanges>  <Rle> | <integer>
##   [1]   chrXIV [304701, 306300]      * |         1
##   [2]   chrXIV [305401, 307000]      * |         2
##   [3]   chrXIV [306401, 308000]      * |         3
##   [4]   chrXIV [306701, 308300]      * |         4
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

2.3.5 How to choose chunk and overhang size

The chunk size shouldn't be below 1kb, but is otherwise unbound. However the bigger the size the longer it will take to compute the model. On the other hand the less chunks the less parallel processes are needed to compute the entire fit. It should also be noted, that a lower number of chunks results in a lower number of overhangs, which are basically a repeated computation of the same region. Thus the total number of basepairs *involved* in computing the model grows with lower chunk size. Since fitting the entire genome as one chunk is not feasible (hence the parallelization approach) there is a trade-off between number of chunks and chunk size. The default value is set to a reasonable size that gave empirically the best runtime results in practice. However this might be different on other systems. Especially if you have little memory, bigger chunks will result in bigger matrices and thus higher memory consumption.

The overhang size is responsible for smooth connection of adjacent chunks. As the fit is dependent on the number of knots and their positions the overhang size is, too. Thus a sufficient number of knots should be selected and the size in basepairs computed from it by multiplying by the knot spacing. In our experience a sufficiently small relative error can already be achieved with around 5-7 knots.

2.3.6 The GenoGAMSettings class

The parameters of the basic functionality of the entire *GenoGAM* pipeline, especially for those steps which involve other R packages, is stored in the *GenoGAMSettings* class. The class has six slots:

- *center*: Set this to FALSE, if you want to count entire fragments instead of only the midpoints.
- *chromosomeList*: If you want to restrict your dataset by chromosome.
- *bamParams*: This is the `ScanBamParam` object from the *Rsamtools* package. It can be useful to restrict the object to a specific set of regions defined through a `GRanges` object.
- *processFunction* Define a different process function for raw data. Not really useful for now.
- *optimMethod* One of the methods from the `optim` function. Applies to the cross validation step.
- *optimControl* A list of control parameters for the `optim` function. Important are the two *maxit* elements. The *maxit* element applies to the number of maximal iterations for the cross validation. The *betaMaxit* element applies to the number of maximal iterations for the beta parameter estimation. Especially with varying chunk sizes (and thus varying number of betas) one might consider modifying the latter.

A couple of examples on how to use the parameters.

```
## change chromosome list
settings <- GenoGAMSettings(chromosomeList = c("chrI", "chrII"))

## This will print out an error from the internal check function.
## But still create an empty object.
temp <- GenoGAMDataSet(
    experimentDesign = expDesign, directory = folder,
    chunkSize = chunkSize, overhangSize = overhangSize,
    design = ~ s(x) + s(x, by = genotype), settings = settings)
```

```
## INFO [2017-05-03 15:58:18] Creating GenoGAMDataSet
## INFO [2017-05-03 15:58:19] Reading in data
## ERROR [2017-05-03 15:58:19] The data does not match the region specification in the bamParams
## INFO [2017-05-03 15:58:19] GenoGAMDataSet created
```

```
temp
```

```
## class: GenoGAMDataSet
## dimension: 0 0
## samples(0):
## design variables(0):
## size factors:
## numeric(0)
## formula:
## ~ s(x)
```

```
## specify parameters through ScanBamParam
gr <- GRanges("chrXIV", IRanges(305001, 308000))
```

```
params <- ScanBamParam(which = gr)
settings <- GenoGAMSettings(bamParams = params)

## read in only the single region we know has counts. No post-subsetting required
temp <- GenoGAMDataSet(
    experimentDesign = expDesign, directory = folder,
    chunkSize = chunkSize, overhangSize = overhangSize,
    design = ~ s(x) + s(x, by = genotype), settings = settings)

## INFO [2017-05-03 15:58:19] Creating GenoGAMDataSet
## INFO [2017-05-03 15:58:19] Reading in data
## INFO [2017-05-03 15:58:19] Reading in wt_1
## INFO [2017-05-03 15:58:19] Reading in wt_2
## INFO [2017-05-03 15:58:19] Reading in mutant_1
## INFO [2017-05-03 15:58:20] Reading in mutant_2
## INFO [2017-05-03 15:58:20] Finished reading in data
## INFO [2017-05-03 15:58:20] GenoGAMDataSet created

temp

## class: GenoGAMDataSet
## dimension: 3000 4
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## design variables(1): genotype
## tiles size: 1.6kbp
## number of tiles: 3
## chromosomes: chrXIV
## size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##        0        0        0        0
## formula:
## ~ s(x) + s(x, by = genotype)

## Count entire fragments
slot(settings, "center") <- FALSE

temp <- GenoGAMDataSet(
    experimentDesign = expDesign, directory = folder,
    chunkSize = chunkSize, overhangSize = overhangSize,
    design = ~ s(x) + s(x, by = genotype), settings = settings)

## INFO [2017-05-03 15:58:20] Creating GenoGAMDataSet
## INFO [2017-05-03 15:58:20] Reading in data
## INFO [2017-05-03 15:58:20] Reading in wt_1
## INFO [2017-05-03 15:58:20] Reading in wt_2
## INFO [2017-05-03 15:58:21] Reading in mutant_1
## INFO [2017-05-03 15:58:21] Reading in mutant_2
## INFO [2017-05-03 15:58:21] Finished reading in data
## INFO [2017-05-03 15:58:21] GenoGAMDataSet created
```

```
## no zero counts anymore due to high fragment coverage
assay(temp)
```

```
## DataFrame with 3000 rows and 4 columns
##       wt_1  wt_2 mutant_1 mutant_2
##      <Rle> <Rle>    <Rle>    <Rle>
## 1        7     6        7        3
## 2        7     6        7        3
## 3        7     6        7        5
## 4        7     6        7        5
## 5        8     6        6        5
## ...    ...   ...      ...      ...
## 2996    18    24       13       21
## 2997    18    23       11       21
## 2998    18    22       11       21
## 2999    18    22       11       21
## 3000    18    22       11       21
```

2.3.7 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. *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 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.

2.4 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:

```
subggd <- computeSizeFactors(subggd)
```

```
## INFO [2017-05-03 15:58:21] Computing size factors
## INFO [2017-05-03 15:58:21] DONE
```

```
sizeFactors(subggd)
```

```
##     wt_1     wt_2 mutant_1 mutant_2
##   0.0122   0.1619  -0.3204   0.3115
```

Note: The size factors in *GenoGAM* are in the natural logarithm scale. Also factor groups are not identified

automatically from the design, but are rather left to the user. By default all samples are treated as members of one factor group and thus size factors are computed together. If custom groups are required, it can be provided separatelly through the *factorGroups* argument.

```
## clear size factors
sizeFactors(subggd) <- rep(0, dim(subggd)[2])
sizeFactors(subggd)
```

```
##     wt_1     wt_2 mutant_1 mutant_2
##        0        0        0        0
```

```
## compute new with factor groups
groups <- list(c("wt_1", "wt_2"), c("mutant_1", "mutant_2"))
subggd <- computeSizeFactors(subggd, factorGroups = groups)
```

```
## INFO [2017-05-03 15:58:21] Computing size factors
## INFO [2017-05-03 15:58:22] DONE
```

```
sizeFactors(subggd)
```

```
##     wt_1     wt_2 mutant_1 mutant_2
##  -0.0597   0.0597  -0.2566   0.2566
```

Note, how the size factors within a group now add up to zero in log space, which is always the case for groups consisting of two elements.

2.5 Model fitting

2.5.1 The `genogam` function

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 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 λ and θ are obtained by cross-validation. This step is a bit time-consuming as it is non-linearly dependent on the data size. That is, the total time for parameter estimation varies much less than the time for model fitting, as the number of regions needed for cross-validation is rather low and at most twenty. Whereas the number of regions for the total model is only bound by genome size. Thus, for sake of going through this example quickly, we provide the values manually. Also we supply the knot spacing at this stage. If it differs from the one used to compute the overhang size, one might end up with too little knots for a sufficient overlap at the junction point of adjacent chunks.

```
## fit model without parameters estimation
fit <- genogam(subggd, lambda = 4601, theta = 4.51, bpknots = bpk)
```

```
## INFO [2017-05-03 15:58:22] Initializing the model
## INFO [2017-05-03 15:58:22] Done
## INFO [2017-05-03 15:58:22] Fitting model
```

```
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## INFO [2017-05-03 15:58:44] Done
## INFO [2017-05-03 15:58:44] Assembling fits and building GenoGAM object
## INFO [2017-05-03 15:58:44] Done
```

```
fit
```

```
## Family:  Negative Binomial
## Formula:  ~ s(x) + s(x, by = genotype)
## Class: GenoGAM
## Dimension: 3000 4
## Samples(4): wt_1 wt_2 mutant_1 mutant_2
## Design variables(1): genotype
## Smooth functions(2): s(x) s(x):genotype
## Chromosomes: chrXIV
##
## Size factors:
##     wt_1     wt_2 mutant_1 mutant_2
##  -0.0597   0.0597  -0.2566   0.2566
##
## Cross Validation:  Not performed
##
## Spline Parameters:
##   Knot spacing:  20
##   B-spline order:  2
##   Penalization order:  2
##
## Tile settings:
##   Chunk size: 1000
##   Tile size: 1000
##   Overhang size: 0
##   Number of tiles: 3
##   Evaluated genome ranges:
## GRanges object with 1 range and 0 metadata columns:
##       seqnames           ranges strand
##          <Rle>        <IRanges>  <Rle>
##   [1]   chrXIV [305001, 308000]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

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. Doing it for a default of 20 different regions (which is the same as tiles in this case). Here, for a tile size of 1.6kb this means that estimation of the parameters will require the equivalent of a *GenoGAM* fit with fixed λ and θ on 16 Mb (1.6kb x10x20x50). For a genome like yeast (12Mb) the cross-validation thus would take more time than a genome-wide fit. In order to avoid this the number of regions is lowered, such that on average cross-validation does not take longer than model fitting. In practice it this will never be triggered unless for small organisms, where even a lower number of regions would be sufficient.

```
fit_CV <- genogam(subggd, 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.

2.5.2 Fitting in case of sparse data

A common problem, especially for larger organisms, is the sparsity of data due to low sequencing depth. If one would attempt to compute a genogam model fit on a large enough region with zero counts, the model would produce artifacts. An easy workaround was suggested by Wood et. al [2]: Adding the identity matrix multiplied by a small enough ϵ to the penalization matrix in order to penalize the absolute counts. If ϵ is small enough it should shrink zero count regions to zero, while having little effect on the other regions. Hence getting rid of the artifacts and stabilizing the fitting process. In order to make use of this in `genogam`, use the *H* parameter, which represents the $epsilon$ value, to pass a small enough value to the penalization matrix. It is zero by default and thus only the second differences are penalized.

```
## a small H shrinks the fit a little bit
fitHsmall <- genogam(subggd, lambda = 4601, theta = 4.51, bpknots = bpk, H = 0.00001)
```

```
## INFO [2017-05-03 16:22:16] Initializing the model
## INFO [2017-05-03 16:22:16] Done
## INFO [2017-05-03 16:22:16] Fitting model
## INFO [2017-05-03 16:22:28] Done
## INFO [2017-05-03 16:22:28] Assembling fits and building GenoGAM object
## INFO [2017-05-03 16:22:28] Done
```

```
## compare with with small H
fits(fitHsmall)
```

```
## DataFrame with 3000 rows and 2 columns
##           s(x) s(x):genotype
##      <numeric>     <numeric>
## 1        -2.76       -0.0636
## 2        -2.76       -0.0620
## 3        -2.76       -0.0605
## 4        -2.76       -0.0589
## 5        -2.76       -0.0574
## ...        ...           ...
## 2996     -1.72       -0.0731
## 2997     -1.71       -0.0752
```

```
## 2998     -1.71       -0.0774
## 2999     -1.71       -0.0795
## 3000     -1.71       -0.0816
```

```
## and fit without H
fits(fit)
```

```
## DataFrame with 3000 rows and 2 columns
##           s(x) s(x):genotype
##      <numeric>     <numeric>
## 1        -2.87        0.0463
## 2        -2.87        0.0478
## 3        -2.87        0.0494
## 4        -2.87        0.0509
## 5        -2.87        0.0524
## ...        ...           ...
## 2996     -1.76       -0.0210
## 2997     -1.76       -0.0232
## 2998     -1.76       -0.0253
## 2999     -1.75       -0.0275
## 3000     -1.75       -0.0296
```

```
## if H is to big, the fit will shrink towards zero
fitH <- genogam(subggd, lambda = 4601, theta = 4.51, bpknots = bpk, H = 0.8)
```

```
## INFO [2017-05-03 16:22:28] Initializing the model
## INFO [2017-05-03 16:22:28] Done
## INFO [2017-05-03 16:22:28] Fitting model
## INFO [2017-05-03 16:22:31] Done
## INFO [2017-05-03 16:22:31] Assembling fits and building GenoGAM object
## INFO [2017-05-03 16:22:31] Done
```

```
fits(fitH)
```

```
## DataFrame with 3000 rows and 2 columns
##           s(x) s(x):genotype
##      <numeric>     <numeric>
## 1     -0.00455      -0.00234
## 2     -0.00470      -0.00241
## 3     -0.00484      -0.00248
## 4     -0.00497      -0.00255
## 5     -0.00511      -0.00262
## ...        ...           ...
## 2996  -0.00447      -0.00232
## 2997  -0.00435      -0.00226
## 2998  -0.00423      -0.00220
## 2999  -0.00411      -0.00213
## 3000  -0.00399      -0.00207
```

2.5.3 The GenoGAM class

The results are stored in the *GenoGAM* class. This class is build, just like *GenoGAMDataSet*, on *RangedSummarizedExperiment*. As such all respective methods are applicable, with a few exceptions:

- `getSettings` now gives access to the settings slot
- `getFamily` provides access to the distribution used
- `getParams` provides access to all parameters used during computation as well as the tile settings
- `fits` and `se` return the pointwise fits and standard errors, respectively

```
## the model formula
design(fit)

## ~s(x) + s(x, by = genotype)

## the size factors
sizeFactors(fit)

##     wt_1     wt_2 mutant_1 mutant_2
##  -0.0597   0.0597  -0.2566   0.2566

## the global model settings
getSettings(fit)

## -------------------- Read-in parameters -----------------
## Center: TRUE
## Chromosomes: chrXIV
## Custom process function: disabled
##
## -------------------- BAM parameters ---------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat: pos, qwidth
## bamMapqFilter: NA
##
## -------------------- Parallel backend -------------------
## 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
##
## -------------------- Optimization parameters ------------
```

```
## Optimization method: Nelder-Mead
## Optimization control:
##   maxit: 50
##   fnscale: -1
##   trace: 1
##   betaMaxit: 2000

## the distribution used
getFamily(fit)

## [1] "nb"

## the parameters used in the model and dataset
getParams(fit)

## $lambda
## [1] 4601
##
## $theta
## [1] 4.51
##
## $H
## [1] 0
##
## $order
## [1] 2
##
## $penorder
## [1] 2
##
## $cv
## [1] FALSE
##
## $bpknots
## [1] 20
##
## $chromosomes
## GRanges object with 1 range and 0 metadata columns:
##       seqnames           ranges strand
##          <Rle>        <IRanges>  <Rle>
##   [1]   chrXIV [305001, 308000]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
##
## $chunkSize
## [1] 1000
##
## $overhangSize
## [1] 0
```

```
##
## $tileSize
## [1] 1000
##
## $numTiles
## [1] 3

## the assay is actually a list of two elements, the fits and the
## standard errors. But calling just assay returns the fits
assays(fit)

## List of length 2
## names(2): fits se

assay(fit)

## DataFrame with 3000 rows and 2 columns
##           s(x) s(x):genotype
##      <numeric>     <numeric>
## 1        -2.87        0.0463
## 2        -2.87        0.0478
## 3        -2.87        0.0494
## 4        -2.87        0.0509
## 5        -2.87        0.0524
## ...        ...           ...
## 2996     -1.76       -0.0210
## 2997     -1.76       -0.0232
## 2998     -1.76       -0.0253
## 2999     -1.75       -0.0275
## 3000     -1.75       -0.0296

## separate functions are recommended to be used for access
fits(fit)

## DataFrame with 3000 rows and 2 columns
##           s(x) s(x):genotype
##      <numeric>     <numeric>
## 1        -2.87        0.0463
## 2        -2.87        0.0478
## 3        -2.87        0.0494
## 4        -2.87        0.0509
## 5        -2.87        0.0524
## ...        ...           ...
## 2996     -1.76       -0.0210
## 2997     -1.76       -0.0232
## 2998     -1.76       -0.0253
## 2999     -1.75       -0.0275
## 3000     -1.75       -0.0296

se(fit)
```

```
## DataFrame with 3000 rows and 2 columns
##           s(x) s(x):genotype
##      <numeric>     <numeric>
## 1        0.256         0.328
## 2        0.255         0.327
## 3        0.254         0.326
## 4        0.253         0.325
## 5        0.252         0.324
## ...        ...           ...
## 2996     0.179         0.236
## 2997     0.180         0.237
## 2998     0.181         0.238
## 2999     0.182         0.239
## 3000     0.183         0.240
```

```
## positions
rowRanges(fit)
```

```
## GPos object with 3000 positions and 0 metadata columns:
##          seqnames       pos strand
##             <Rle> <integer>  <Rle>
##      [1]   chrXIV    305001      *
##      [2]   chrXIV    305002      *
##      [3]   chrXIV    305003      *
##      [4]   chrXIV    305004      *
##      [5]   chrXIV    305005      *
##      ...      ...       ...    ...
##   [2996]   chrXIV    307996      *
##   [2997]   chrXIV    307997      *
##   [2998]   chrXIV    307998      *
##   [2999]   chrXIV    307999      *
##   [3000]   chrXIV    308000      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## subsetting works just like in GenoGAMDataSet
subfit <- subset(fit, pos < 306001)
rowRanges(subfit)
```

```
## GPos object with 1000 positions and 0 metadata columns:
##          seqnames       pos strand
##             <Rle> <integer>  <Rle>
##      [1]   chrXIV    305001      *
##      [2]   chrXIV    305002      *
##      [3]   chrXIV    305003      *
##      [4]   chrXIV    305004      *
##      [5]   chrXIV    305005      *
##      ...      ...       ...    ...
##    [996]   chrXIV    305996      *
```

```
##    [997]   chrXIV    305997      *
##    [998]   chrXIV    305998      *
##    [999]   chrXIV    305999      *
##   [1000]   chrXIV    306000      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
gr <- GRanges("chrXIV", IRanges(306001, 307000))
subfit <- fit[gr]
rowRanges(subfit)
```

```
## GPos object with 1000 positions and 0 metadata columns:
##          seqnames       pos strand
##             <Rle> <integer>  <Rle>
##      [1]   chrXIV    306001      *
##      [2]   chrXIV    306002      *
##      [3]   chrXIV    306003      *
##      [4]   chrXIV    306004      *
##      [5]   chrXIV    306005      *
##      ...      ...       ...    ...
##    [996]   chrXIV    306996      *
##    [997]   chrXIV    306997      *
##    [998]   chrXIV    306998      *
##    [999]   chrXIV    306999      *
##   [1000]   chrXIV    307000      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

3 Acknowledgments

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

4 Session Info

- R version 3.3.3 (2017-03-06), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `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, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, Biostrings 2.42.1, fastGenoGAM 1.99, GenomeInfoDb 1.10.3, GenomicRanges 1.26.4, IRanges 2.8.2, knitr 1.15.1, Rsamtools 1.26.2, S4Vectors 0.12.2, SummarizedExperiment 1.4.0, XVector 0.14.1
- Loaded via a namespace (and not attached): acepack 1.4.1, annotate 1.52.1, AnnotationDbi 1.36.2, backports 1.0.5, base64enc 0.1-3, BiocParallel 1.8.2, bitops 1.0-6, checkmate 1.8.2, cluster 2.0.6,

codetools 0.2-15, colorspace 1.3-2, data.table 1.10.4, DBI 0.6-1, DESeq2 1.14.1, digest 0.6.12, evaluate 0.10, foreign 0.8-67, Formula 1.2-1, futile.logger 1.4.3, futile.options 1.0.0, genefilter 1.56.0, geneplotter 1.52.0, GenomicAlignments 1.10.1, ggplot2 2.2.1, grid 3.3.3, gridExtra 2.2.1, gtable 0.2.0, HDF5Array 1.2.1, highr 0.6, Hmisc 4.0-2, htmlTable 1.9, htmltools 0.3.5, htmlwidgets 0.8, lambda.r 1.1.9, lattice 0.20-35, latticeExtra 0.6-28, lazyeval 0.2.0, locfit 1.5-9.1, magrittr 1.5, Matrix 1.2-8, memoise 1.0.0, munsell 0.4.3, nnet 7.3-12, plyr 1.8.4, RColorBrewer 1.1-2, Rcpp 0.12.10, RCurl 1.95-4.8, rhdf5 2.18.0, rpart 4.1-10, RSQLite 1.1-2, scales 0.4.1, splines 3.3.3, stringi 1.1.5, stringr 1.2.0, survival 2.41-3, tibble 1.3.0, tools 3.3.3, XML 3.98-1.6, xtable 1.8-2, zlibbioc 1.20.0

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.

[2] Simon Wood. *Generalized additive models: an introduction with R*. CRC press, 2006.