Stricker *et al. BMC Bioinformatics* (2018) 19:247
https://doi.org/10.1186/s12859-018-2238-7

BMC Bioinformatics

SOFTWARE Open Access

GenoGAM 2.0: scalable and efficient implementation of genome-wide generalized additive models for gigabase-scale genomes

Georg Stricker, Mathilde Galinier and Julien Gagneur*

Abstract

Background: GenoGAM (Genome-wide generalized additive models) is a powerful statistical modeling tool for the analysis of ChIP-Seq data with flexible factorial design experiments. However large runtime and memory requirements of its current implementation prohibit its application to gigabase-scale genomes such as mammalian genomes.

Results: Here we present GenoGAM 2.0, a scalable and efficient implementation that is 2 to 3 orders of magnitude faster than the previous version. This is achieved by exploiting the sparsity of the model using the SuperLU direct solver for parameter fitting, and sparse Cholesky factorization together with the sparse inverse subset algorithm for computing standard errors. Furthermore the HDF5 library is employed to store data efficiently on hard drive, reducing memory footprint while keeping I/O low. Whole-genome fits for human ChIP-seq datasets (ca. 300 million parameters) could be obtained in less than 9 hours on a standard 60-core server. GenoGAM 2.0 is implemented as an open source `R` package and currently available on GitHub. A Bioconductor release of the new version is in preparation.

Conclusions: We have vastly improved the performance of the GenoGAM framework, opening up its application to all types of organisms. Moreover, our algorithmic improvements for fitting large GAMs could be of interest to the statistical community beyond the genomics field.

Keywords: Genome-wide analysis, ChIP-Seq, Generalized additive models, Sparse inverse subset algorithm, Transcription factors

Background

Chromatin immunoprecipitation followed by deep sequencing (ChIP-Seq), is the reference method for quantification of protein-DNA interactions genome-wide [1, 2]. ChIP-Seq allows studying a wide range of fundamental cellular processes such as transcription, replication and genome maintenance, which are characterized by occupancy profiles of specific proteins along the genome. In ChIP-Seq based studies, the quantities of interest are often the differential protein occupancies between experiments and controls, or between two genetic backgrounds, or between two treatments, or combinations thereof.

We have recently developed a statistical method, GenoGAM (Genome-wide Generalized Additive Model), to flexibly model ChIP-Seq factorial design experiments [3]. GenoGAM models ChIP-Seq read count frequencies as products of smooth functions along chromosomes. It provides base-level and region-level significance testing. An important advantage of GenoGAM over competing methods is that smoothing parameters are objectively estimated from the data by cross-validation, eliminating ad-hoc binning and windowing. It leads to increased sensitivity in detecting differential protein occupancies over competing methods, while controlling for type I error rates.

GenoGAM is implemented as an `R` package based on the well-established and flexible generalized additive models (GAM) framework [4]. On the one hand, it builds on top of the infrastructure provided by the *Bioconductor* software project [5]. On the other hand, it uses the `mgcv` package [6], a general-purpose `R` library for fitting GAMs [7] that provides a rich functionality for GAMs with a variety

*Correspondence: gagneur@in.tum.de
Department of Informatics, Technical University Munich, Boltzmannstr. 3, Garching, Germany

© The Author(s). 2018 **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The Creative Commons Public Domain Dedication waiver (http://creativecommons.org/publicdomain/zero/1.0/) applies to the data made available in this article, unless otherwise stated.

of basis functions, distributions and further features for variable and smoothness selection. In its general form, the implementation for fitting a GAM minimizes a cost function using iterations whose time complexity are quadratic in the number of parameters. Moreover, the time complexity of the implementation for estimating the standard errors of the parameters, which are required for any statistical significance assessment, is cubic in the number of parameters. To allow the fitting of GAMs on complete genomes, which involves millions of parameters, we had proceeded with a tiling approach [3]. Genome-wide fits were obtained by fitting models on *tiles*, defined as overlapping genomic intervals of a tractable size, and joining together tile fits at overlap midpoints. With long enough overlaps, this approximation yielded computation times linear in the number of parameters at no practical precision cost. Furthermore, it allowed for parallelization, with speed-ups being linear in the number of cores.

Nonetheless, application of the current implementation remains limited in practice to small genomes organisms such as yeast or bacteria, or to selected subsets of larger genomes. A genome-wide fit for the yeast genome (ca. 1 million parameters) took 20 hours on a 60-core server. Fits for the human genome could only be done for chromosome 22, the smallest human chromosome.

Here we introduce a new implementation of GenoGAM that is 2 to 3 orders of magnitude faster. This is achieved by exploiting the sparsity of the model and by using out-of-core data processing. The computing time for parameter and standard error estimation, as well as the memory footprint, is now linear in the number of parameters per tile. The same genome-wide fit for yeast is now obtained in 13 min on a standard 8-core desktop machine. Whole-genome fits for human datasets (ca. 300 million parameters each) are obtained in less than 9 hours on the same 60-core server.

Before describing the new implementation and results, we provide some necessary mathematical background.

GenoGAM models

In a GenoGAM model, we assume ChIP-Seq read counts y_i at genomic position x_i in the ChIP-Seq sample j_i to follow a negative binomial distribution with mean μ_i and dispersion parameter θ:

$$y_i \sim \mathrm{NB}(\mu_i, \theta) \tag{1}$$

where the logarithm of the mean μ_i is the sum of an offset o_i and one or more smooth functions f_k of the genomic position x_i:

$$\log(\mu_i) = o_i + \sum_{k=1}^{K} f_k(x_i) z_{j_i,k} \tag{2}$$

The offsets o_i are predefined data-point specific constants that account for sequencing depth variations. The elements $z_{j_i,k}$ of the experimental design matrix $\mathbf{Z}$ is 1 if smooth function f_k contributes to the mean counts of sample j_i and 0 otherwise. A typical application is the comparison of treatment versus control samples, for which a GenoGAM model would read:

$$\log(\mu_i) = o_i + f_{\text{control}}(x_i) + z_{j_i} f_{\text{treatment/control}}(x_i), \tag{3}$$

where $z_{j_i} = 0$ for all control sample data points and $z_{j_i} = 1$ for all treatment sample data points. The quantity of interest in such a scenario is the log fold-change of treatment versus control at every genomic position $f_{\text{treatment/control}}(x_i)$.

The smooth functions f_k are piecewise polynomials consisting of a linear combination of basis functions b_r and the respective coefficients $\beta_r^{(k)}$:

$$f_k(x_i) = \sum_r \beta_r^{(k)} b_r(x_i) := \left(\mathbf{X}_k \boldsymbol{\beta}^{(k)}\right)_i, \tag{4}$$

where b_r are cubic B-splines, which are bell-shaped cubic polynomials over a finite local support [8]. The column of the $n \times p_k$ matrix $\mathbf{X}_k$, where p_k is the number of basis functions in smooth f_k, represents a basis function b_r evaluated at each position x_i.

Typically all smooth functions have the same bases and knot positioning, implying that all $\mathbf{X}_k$ are equal to each other. Consequently, the complete design matrix $\mathbf{X}$ is the Kronecker product of the experimental design matrix $\mathbf{Z}$ and $\mathbf{X}_k$.

$$\log(\mu_i) = o_i + (\mathbf{X}\boldsymbol{\beta})_i, \tag{5}$$

where $\mathbf{X} = \mathbf{Z} \otimes \mathbf{X}_k$ and the vector $\boldsymbol{\beta}$ is the concatenation of all $\boldsymbol{\beta}^{(k)}$.

The fitting of the parameters $\boldsymbol{\beta}$ is carried out by maximizing the negative binomial log-likelihood plus a penalty function:

$$\hat{\boldsymbol{\beta}} = \operatorname{argmax} \left\{ l_{\mathrm{NB}}(\boldsymbol{\beta}; \boldsymbol{y}, \theta) - \lambda \boldsymbol{\beta}^T (\mathbf{S} + \epsilon \mathbf{I}) \boldsymbol{\beta} \right\} \tag{6}$$

where $\mathbf{S}$ is a symmetric positive matrix that approximately penalizes the second order derivatives of the smooth functions. This approach is called penalized B-splines or P-splines [9]. The $\epsilon\mathbf{I}$ term adds regularization on the squared values of the $\boldsymbol{\beta}$'s, which is particularly useful for regions with many zero counts. The smoothing parameter λ controls the amount of regularization. Both the smoothing parameter λ and the dispersion parameter θ are considered as hyperparameters that are estimated by cross-validation [3].

Newton-Raphson methods are used to maximize Eq. (6). The idea is to iteratively maximize quadratic approximations of the objective function around the

current estimate. The current parameter vector $\boldsymbol{\beta}_t$ is updated as:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \mathbf{H}^{-1}(\boldsymbol{\beta}_t)\nabla f(\boldsymbol{\beta}_t) \tag{7}$$

where the negative inverse Hessian $\mathbf{H}^{-1}(\boldsymbol{\beta}_t)$ captures the local curvature of the objective function, and the gradient vector $\nabla f(\boldsymbol{\beta}_t)$ captures the local slope. The iteration stops when the change in the log-likelihood or the norm of the gradient of the log-likelihood falls below a specified convergence threshold. Because the negative binomial distribution with known dispersion parameter θ is part of the exponential family, the penalized log-likelihood is convex and thus convergence is guaranteed.

Standard error computation

For the purpose of statistical testing, variance of the smooth estimates are also needed. These are of the form:

$$\mathrm{Var}(f_k(x_i)) = \sigma^2_{i,k} = \left(\mathbf{X}_k \mathbf{H}_k^{-1} \mathbf{X}_k^T\right)_{i,i}, \tag{8}$$

where $\mathbf{H}_k$ is the Hessian with respect to the parameters $\boldsymbol{\beta}^{(k)}$ and can be simply extracted from $\mathbf{H}$.

Remarks on sparsity

The Hessian $\mathbf{H}$ is computed as:

$$\mathbf{H} = \mathbf{X}^T \mathbf{W} \mathbf{X} - 2\lambda(\mathbf{S} + \epsilon \mathbf{I}) \tag{9}$$

with $\mathbf{W}$ a diagonal matrix [6].

However, the number of nonzeros for each row of the design matrix $\mathbf{X}$ is at most 5 times the number of smooth functions because every genomic position x_i is overlapped by 5 cubic B-splines b_r only. Moreover, the penalization matrix $\mathbf{S}$ only has 5 nonzeros per row, as it encodes the second-order difference penalties between coefficients of neighboring splines [9]. Hence, the matrices $\mathbf{X}$ and $\mathbf{S}$, and therefore $\mathbf{H}$, which appears in the majority of the computations via Eqs. (7) and (8), are very sparse. Here we make use of the sparsity of these matrices to drastically speed up the fitting of the parameters.

Implementation

Workflow

Data preprocessing consists of reading raw read alignments from BAM files, centering the fragments, computing the coverage $\boldsymbol{y}$, and splitting the data by genomic tiles (Fig. 1). Afterwards, normalization factors for sequencing depth variation are computed using DESeq2 [10]. In the new version of GenoGAM we store the preprocessed data in HDF5 files [11] through the `R` packages `HDF5Array` [12] and `rhdf5` [13]. This allows writing in parallel as the data is being preprocessed, which reduces the memory footprint of this step. For all subsequent matrix operations the `Matrix` package is used, which implements routines for storage, manipulation and operations on sparse matrices [14].

Fitting GenoGAM models on tiles is achieved by the Newton-Raphson algorithm (Eq. 7). This is done on few representative tiles during cross-validation in order to identify optimal hyperparameters λ and θ, and subsequently when fitting the model on the full dataset.

The variance of the smooth estimates (Eq. 8) is obtained using the sparse inverse subset algorithm as detailed in a subsection below. The implementation is based on the `R` package `sparseinv` [15], which wraps relevant code from the `SuiteSparse` software [16]. As in the previous GenoGAM model [3], fitting on different tiles is conducted in parallel. The result objects for the fits, variances and parameters are initialized prior to fitting on hard drive. This allows the processes to write results in parallel on the fly, ensuring fast computation and low memory footprint. The HDF5 storage is further optimized for reading time by adjusting HDF5 chunk size to the size of the tiles (for preprocessed count data) and chunks (for fits and variance). As HDF5 is not process-safe on `R` level, writing is serialized by a queuing mechanism.

The parallelization backend is provided by the `R` package `BiocParallel`. It offers an interface to a variety of backends and can be registered independently of GenoGAM. Parallelization is performed over chromosomes during the read-in process. Over tuples of folds and tiles during cross-validation process and over tiles during fitting process. Because some backends have a particular long start-up time, the use of many processes might end up dominating computation time. Specifically during cross-validation on small and limited number of regions, this might pose a problem. Therefore an optimal number of workers is automatically obtained and registered by the cross-validation function and reset on exit.

Newton-Raphson implementation for sparse matrices

We estimate the parameters $\boldsymbol{\beta}$ by maximizing the penalized log-likelihood using the Newton-Raphson iteration (Eq. 7). Due to the sparsity of the matrices $\mathbf{X}$, $\mathbf{D}$ and $\mathbf{S}$, $\mathbf{H}$ is sparse and cheap to compute. The inverse is never explicitly formed. Instead the linear system is solved by a direct solver using the `SuperLU` library [17]. Furthermore all matrices are stored in a sparse format, avoiding redundant storage of zeros.

Our new fitting algorithm differs from the one of `mgcv` in two ways. First, `mgcv` uses Iteratively Reweighted Least Squares, a Newton-Raphson method that employs the Fisher information matrix $\mathcal{I}$, defined as the negative expectation of the Hessian $\mathbf{H}$, instead of the Hessian in the iteration (Eq. 7). However, this did not lead to any measurable differences in the fitted parameters. Second, `mgcv` uses QR decomposition of the design matrix $\mathbf{X}$ [6]. However, general QR decomposition destroys the sparse

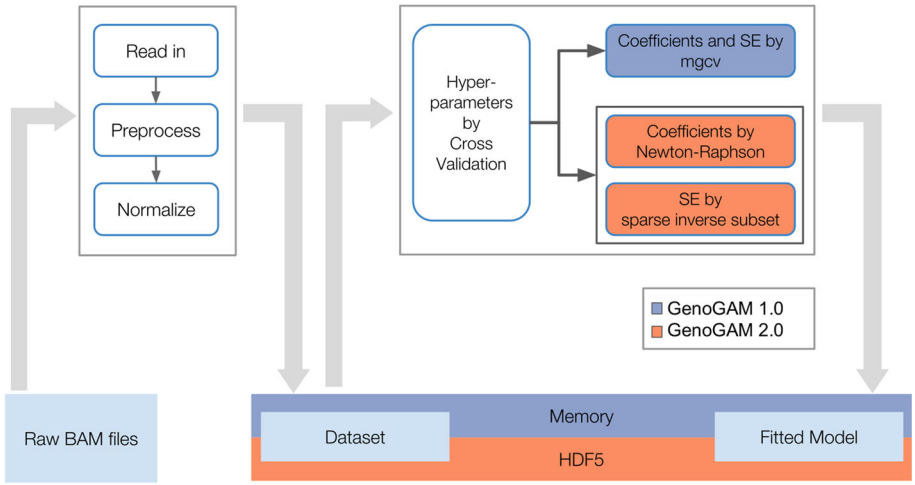

Fig. 1 Schematic overview highlighting the difference between GenoGAM 1.0 and GenoGAM 2.0: Raw BAM Files are read-in, pre-processed normalized and written to hard drive in HDF5 format. Moreover, normalization factors for sequencing depth variation are computed using DESeq2 [10]. The resulting object is the dataset upon which fitting is done. Then global hyperparameters are estimated by cross-validation and for each tile coefficients are estimated via Newton-Raphson and standard errors via sparse inverse subset algorithm. The final model is written as a new object to hard drive in HDF5 format. Note, that the schematic view is a simplification: The pre-processed dataset and the fitted model are not generated in memory and written to HDF5 in the end. Instead, all HDF5 matrices are initialized on hard drive directly and the writing is done on the fly. Blue (GenoGAM 1.0) and orange colors (GenoGAM 2.0) mark differences between both GenoGAM versions, simultaneously displaying the content of this paper

structure of $\mathbf{X}$. We have investigated the use of sparse QR decompositions but this was less efficient than our final implementation.

Variance computation using the sparse inverse subset algorithm

The Hessian $\mathbf{H}$ is sparse, but its inverse, the covariance matrix $\mathbf{H}^{-1}$, usually is not. However, the variances of interest (Eq. 8) can be computed using only a subset of the elements of the inverse $\mathbf{H}^{-1}$. Specifically, denoting for any matrix $\mathbf{A}$:

- $\mathrm{NZ}(\mathbf{A}) = \{(i,j), \mathbf{A}_{i,j} \neq 0\}$ the indices of nonzero elements,
- $C_i(\mathbf{A}) = \{j : \mathbf{A}_{i,j} \neq 0\}$ the column indices of nonzero elements for the i-th row,
- $R_j(\mathbf{A}) = \{i : \mathbf{A}_{i,j} \neq 0\}$ the column indices of nonzero elements for the j-th row,

then $\boldsymbol{\sigma}^2$ can be computed only using the elements $(\mathbf{H}^{-1})_{l,j}$, where $(l,j) \in \mathrm{NZ}(\mathbf{H})$. Indeed, on the one hand we have:

$$\begin{aligned}\sigma_i^2 &= \sum_{l,j} \mathbf{X}_{i,l} \left(\mathbf{H}^{-1}\right)_{l,j} \mathbf{X}_{i,j} \\ &= \sum_{(l,j)\in C_i^2(\mathbf{X})} \mathbf{X}_{i,l} \left(\mathbf{H}^{-1}\right)_{l,j} \mathbf{X}_{i,j}\end{aligned} \tag{10}$$

On the other hand, Eq. 9 implies that $\mathrm{NZ}(\mathbf{H}) = \mathrm{NZ}(\mathbf{X}^T\mathbf{W}\mathbf{X}) \cup \mathrm{NZ}(\mathbf{S}) \cup \mathrm{NZ}(\mathbf{I})$. Since

$$(\mathbf{X}^T\mathbf{W}\mathbf{X})_{l,j} = \left(\sum_i \mathbf{X}_{i,l}\mathbf{W}_{i,i}\mathbf{X}_{i,j}\right), \tag{11}$$

it follows that:

$$\begin{aligned}\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)_{l,j} \neq 0 &\Leftrightarrow \exists i, i \in R_l(\mathbf{X}) \text{ and } i \in R_j(\mathbf{X}) \\ &\Leftrightarrow \exists i, (l,j) \in C_i^2(\mathbf{X})\end{aligned}$$

Moreover, the nonzeros of the identity matrix $\mathbf{I}$ is a subset of the nonzeros of the second-order differences penalization matrix $\mathbf{S}$ [9]. Furthermore, the nonzeros of the second-order differences penalization matrix $\mathbf{S}$, which penalizes differences between triplets of consecutive splines, is a subset of the nonzeros of $\mathbf{X}^T\mathbf{X}$, since genomic positions overlap five consecutive splines when using cubic B-splines. Hence, $\mathrm{NZ}(\mathbf{H}) = \left\{(l,j), \exists i, (l,j) \in C_i^2(\mathbf{X})\right\}$. Together with Eq. 10, this proves the result.

Using only the elements of $\mathbf{H}^{-1}$ that are in $\mathrm{NZ}(\mathbf{H})$ applies to computing the variance of any linear combinations of the $\boldsymbol{\beta}$ based on the same sparse structure of $\mathbf{X}$ or a subset of it. Hence, it applies to computing the variance of the predicted value for any smooth function $f_k(x)$ or computing the variance of the derivatives of any order r of any smooth $\frac{d^r f_k(x)}{d^r x}$.

To obtain the elements of $\mathbf{H}^{-1}$ that are in $\mathrm{NZ}(\mathbf{H})$, we used the sparse inverse subset algorithm [18]. Given a

sparse Cholesky decomposition of symmetric matrix $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, the sparse inverse subset algorithm returns the values of the inverse $\mathbf{A}^{-1}$ that are nonzero in the Cholesky factor $\mathbf{L}$. Since nonzero in the lower triangle of $\mathbf{A}$ are nonzeros in the Cholesky factor $\mathbf{L}$ [19], the sparse inverse subset algorithm provides the required elements of $\mathbf{H}^{-1}$ when applied to a sparse Cholesky decomposition of $\mathbf{H}$. See also Rue [20] for similar ideas for Gaussian Markov Fields. To perform the sparse inverse subset algorithm, we used the `R` package `sparseinv` [15], itself a wrapper of relevant code from the `SuiteSparse` software [16].

Once the sparse inverse subset of the Hessian is obtained, σ_i^2 can be computed according to Eq. (10) with a slight improvement: Because only the diagonal is needed from the final matrix product, the implementation does not perform two matrix multiplications. Instead, only the first product is computed, then multiplied element-wise with $\mathbf{X}_k^T$ and summed over the columns.

Results

Leveraging the sparse data structure allows for faster parameter estimation

Figure 2 displays the comparison in fitting runtime (A) and memory usage (B) of our Newton-Raphson method versus the method underlying the previous GenoGAM version on a single core. Computation was capped at approximately 2 h, which leads the blue line (GenoGAM 1.0) to end after around 1100 parameters. It can be clearly seen that exploiting the advantages of the data structures leads to improvements by 2 to 3 orders of magnitude. At the last comparable point at 1104 parameters it took the previous method 1 hours and 37 min, while it was only 1 s for the Newton-Raphson method. This number increased a little bit towards the end to almost 5 s for 5000 parameters.

Additionally, the more efficient storage of sparse matrices and the lightweight implementation reduces the overhead and memory footprint. Again at the last comparable point, the memory used by the previous method is 8 Gbyte while it is 52 MByte by the new method, increasing to 250 MByte at the 5000 parameters mark. Moreover, runtime per tile drops empirically from growing cubically with the number of parameters in GenoGAM 1.0 to linearly in GenoGAM 2.0. Also, The memory footprint drops empirically from growing quadratically with the number of parameters in GenoGAM 1.0 to linearly in GenoGAM 2.0 (dashed black lines fitted to the performance data).

Exact σ^2 computation by the sparse inverse subset algorithm

Alternatively to the direct computation of the inverse Hessian with consecutive computation of variance vector $\boldsymbol{\sigma}^2$,

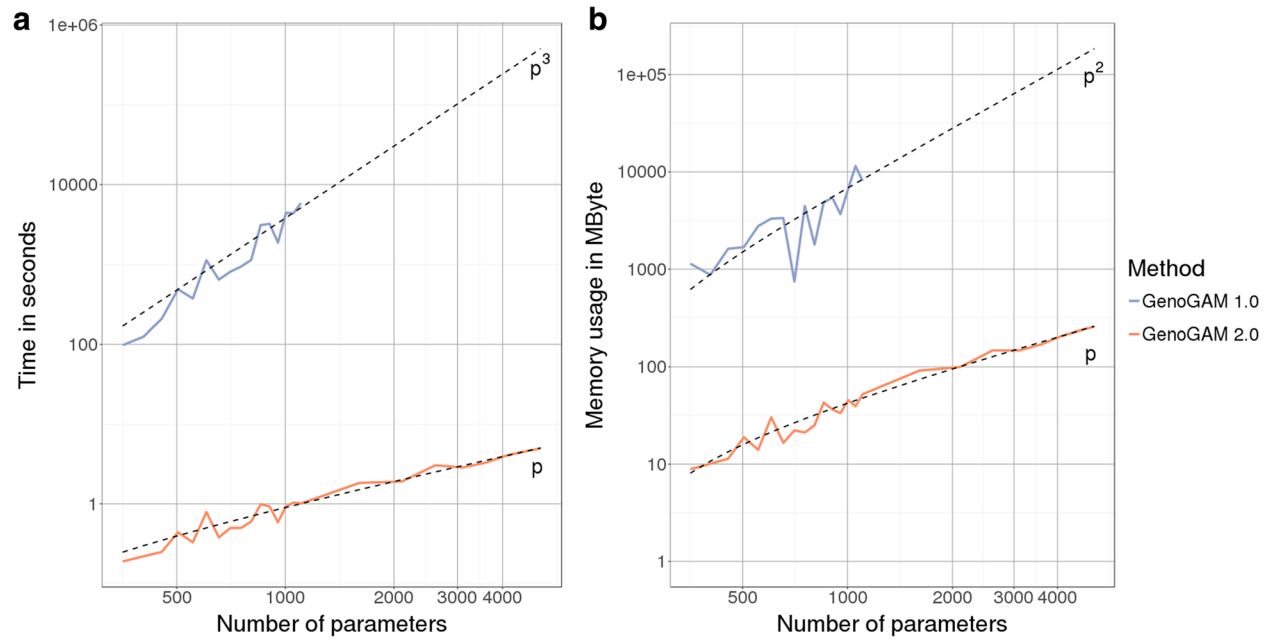

Fig. 2 Coefficient estimation performance. **a** Empirical runtime for the estimation of coefficients vector $\boldsymbol{\beta}$ is plotted in log-scale against increasing number of parameters (also log-scale). The runtime is capped at around 2 hours, such that runtime of previous GenoGAM version (blue line) terminates after 1100 parameters. The new version of GenoGAM (orange line) achieves linear runtime in p (dotted line p), the number of parameters, compared to the previous cubic complexity (dotted line p^3). **b** Memory consumption in MByte for the estimation of coefficients vector $\boldsymbol{\beta}$ is plotted against number of parameters (also log-scale). Due to the runtime cap at around 2 hours the runtime of previous GenoGAM version (blue line) does terminate after 1100 parameters. The storage of matrices in sparse format and direct solvers avoiding full inversion keep the memory footprint low and linear in p (dotted line p) for the new GenoGAM version (orange line) compared to quadratic in the previous version (blue line, dotted line p^2)

it is also possible to directly compute $\boldsymbol{\sigma}^2$. Here and hereafter the smooth function specific index k is dropped for simplicity. In a comment to the paper of Lee and Wand [21], a direct way to compute $\boldsymbol{\sigma}^2$ without inverting $\mathbf{H}$ was proposed by Simon Wood [22]. The comment states, that in general, if $y = \mathbf{X}\beta$, then

$$\sigma_i^2 = \sum_{j=1}^{p} \left(\left(\mathbf{X}\mathbf{P}^T\mathbf{L}^{-1} \right)_{i,j} \right)^2 \tag{12}$$

Where $\mathbf{P}$ is the permutation matrix and $\mathbf{L}^{-1}$ is the inverted lower triangular matrix resulting from Cholesky decomposition of $\mathbf{X}^T\mathbf{H}^{-1}\mathbf{X}$.

Figure 3 shows the comparison of both methods in time and memory on a single core, with the above proposed method depicted as "indirect" (blue). While both methods have linear memory footprint, the slope of the indirect method is around four times higher. The computation time is significantly in favor of the sparse inverse algorithm. This is because for every σ_i^2 a triangular system has to be solved to obtain $(\mathbf{X}\mathbf{P}^T)_i\mathbf{L}^{-1}$. Although solving the complete system at once is faster, it had a high memory consumption when it came to increased number of parameters in our implementation. Thus the performance presented is based on batches of σ_i^2 to obtain a fair trade-off between runtime and memory. Nevertheless, the difference remains around 2 orders of magnitude. Moreover runtime goes now linearly in practice for the sparse inverse subset algorithm compared to quadratically for the indirect method (dashed black lines fitted to the performance data).

Performance on human and yeast ChIP-Seq datasets

The previous version of GenoGAM could only be partially applied genome-wide for megabase-scale genomes such as the yeast genome and was impractical for gigabase-scale genomes such as the human genome. A genome-wide model fit with two conditions and two replicates each took approximately 20 h on 60 cores [3]. With computational and numerical improvement on one side and a data model largely stored on hard drive on the other side, runtime and memory requirements have dropped significantly. Figure 4 shows the runtime performance on seven human ChIP-Seq datasets with two replicates for the IP and one or two replicates for the control. The analysis was performed with 60 cores on a cluster, the memory usage never exceeded 1.5 GB per core and was mostly significantly lower. The overall results show that around 20 min are spent with pre-processing the data, which is largely occupied by writing the data to HDF5

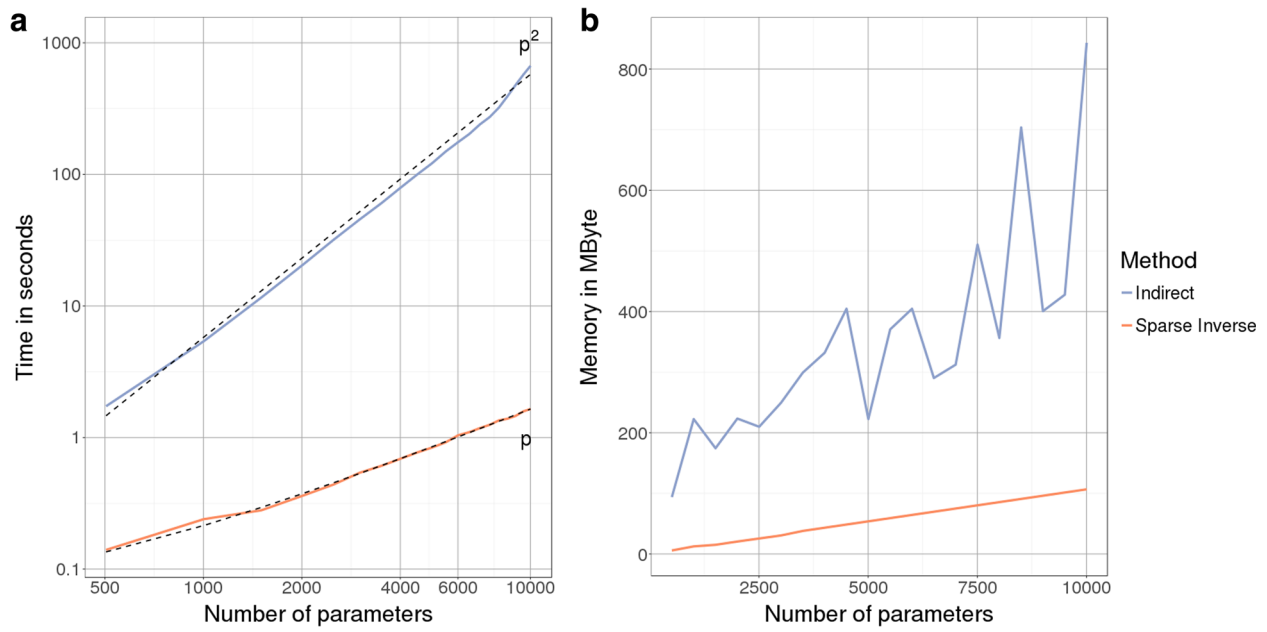

Fig. 3 Standard error computation. **a** Empirical runtime for the computation of standard error vector $\boldsymbol{\sigma}^2$ is plotted in log-scale against increasing number of parameters (also log-scale). Computation based on sparse inverse subset algorithm (orange line) achieves linear runtime in p (dotted line p), the number of parameters, compared to quadratic complexity (dotted line p^2) of the "indirect" method (blue line). **b** Memory consumption in MByte for the computation of standard error vector $\boldsymbol{\sigma}^2$ is plotted against number of parameters. Though both methods achieve linear memory consumption in p, the slope of the "indirect" method (blue line) is around 4 times greater than of the sparse inverse subset algorithm (orange line) likely due to the recursive computation of the inverse instead of solving of a triangular system

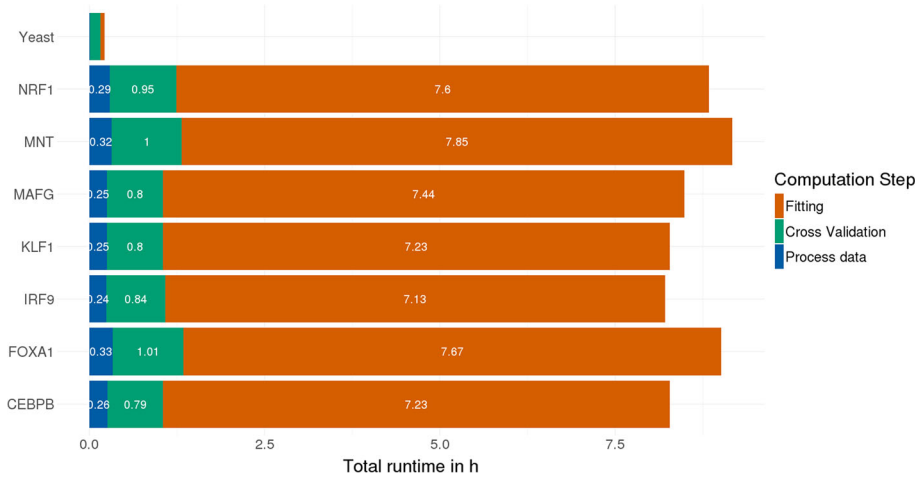

Fig. 4 Genome-wide performance for human and yeast. The performance of GenoGAM 2.0 on seven human ChIP-Seq datasets for the transcription factors NRF1, MNT, FOXA1, MAFG, KLF1, IRF9 and CEBPB. The first three of which contain two replicates for the control, while the rest contains only one. This increases the data by around a 1/3, but the runtime by around 40 min, equivalent to approximately 1/11. Overall ca. 20 min are spent on data processing (blue), up to one hour on cross-validation (green) and 7 - 8 h of fitting (orange) amounting to a total of 8 - 9 h runtime on 60 cores, with the `snow` parallel backend and HDF5 data structure. At the very top yeast runtime is shown on a regular machine with 8 cores, the `multicore` backend and all data kept in memory avoiding I/O to hard drive. Data processing (blue, almost not visible) takes 40 s, cross-validation around 9 min (green) and fitting 3.5 min (red)

files. One hour of cross-validation, to find the right hyperparameters and around 7 to 8 h of fitting, amounting to a total runtime of 8 to 9 h. It is also notable, that the transcription factors NRF1, MNT and FOXA1 include two controls instead of one, thus efficiently increasing the amount of data to fit by a third, but the runtime by around 40 min.

Additionally, the same yeast analysis is shown by running on a laptop with 8 cores for comparison to the previous version. The total runtime is around 13 min with the cross-validation significantly dominating both other steps (around 9 min). This is due to the fact, that the number of regions used is fixed at 20, resulting in 200 model fitting runs for one 10-fold cross-validation iteration. Hence, for a small genome like the yeast genome, hyperparameter optimization may take more time than the actual model fitting. Note, that during cross-validation the only difference between human and yeast analysis is the underlying data and the parallel backend. However the runtime on yeast is only 1/6 of the runtime in human. Both factors play a role in this: First, the parallel backend in the yeast run uses the `Multicore` backend, allowing for shared memory on one machine. While the human run uses the `Snow` (**s**imple **n**etwork **o**f **w**orkstations) backend, which needs to initiate the workers and copy the needed data first, resulting in an overall greater overhead. Second, convergence on yeast data is generally faster due to higher coverage resulting not only in less iterations by the Newton-Raphson, but also during cross-validation.

Replication of previous benchmark analyses show equivalent biological accuracy

To demonstrate that GenoGAM 2.0 leads to the same results than GenoGAM 1.0 we have re-generated benchmark analyses of the first paper [3]. The first benchmark is a differential occupancy application that demonstrates that GenoGAM has greater sensitivity for same specificities than alternative methods (Fig. 5a-b). The second benchmark shows that GenoGAM is on par with alternative methods to infer peak summit positions in ChIP-Seq data of transcription factors (Fig. 5c). Consistently, with the fact that GenoGAM 2.0 fits the same function than GenoGAM 1.0, the performance on these two benchmarks matched.

These improvements have required us to re-implement the fitting of generalized additive models, since GenoGAM 1.0 was based on an generic R package for fitting generalized additive models. We have restricted the implementation so far to the negative binomial distribution. Therefore, application to methylation data, which requires the quasi-binomial distribution, is not yet supported.

Conclusion

We have significantly improved the implementation of GenoGAM [3] on three main aspects: Data storage, coefficient estimation and standard error computation. We showed its runtime and memory footprint to scale linearly with the number of parameters per tiles. As a result, GenoGAM can be applied overnight to gigabase-scale

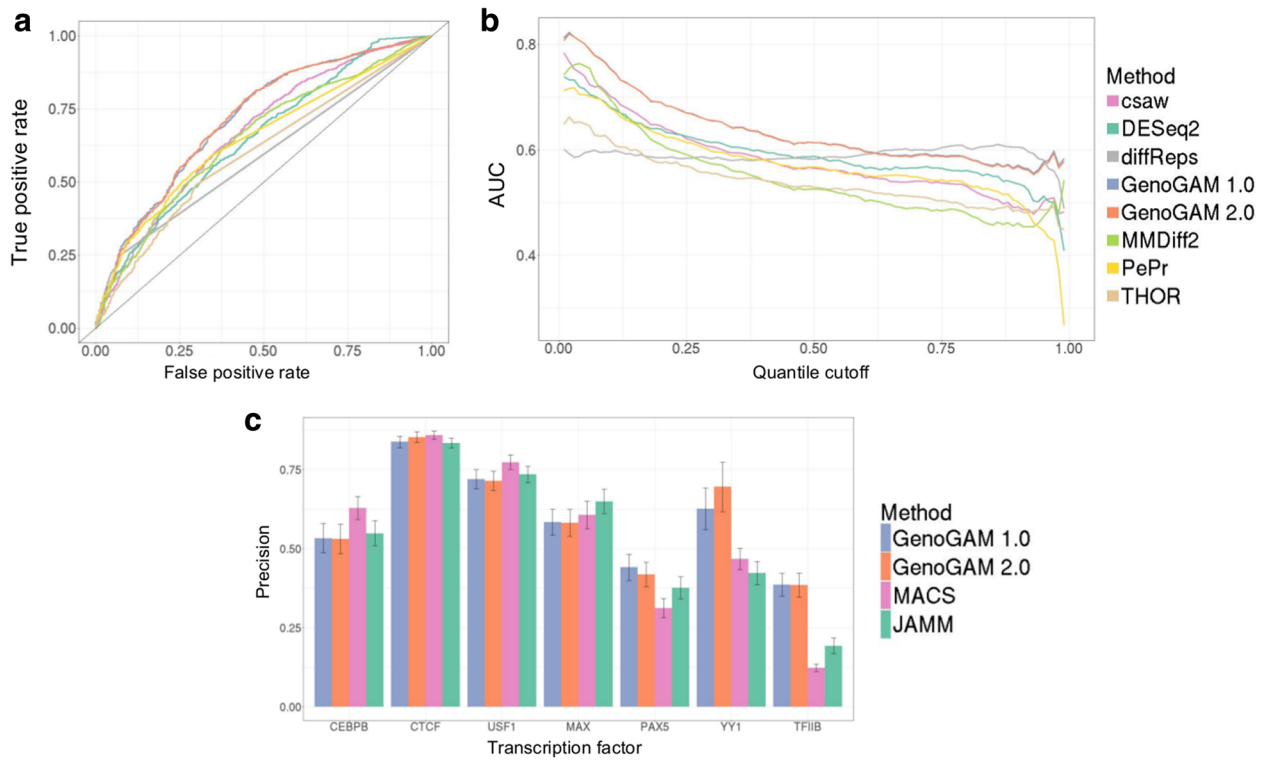

Fig. 5 Replication from our previous study [3] with GenoGAM 2.0. **a** Replication of figure 3A from our previous study [3]. ROC curve based on a quantile cutoff of 0.15 (15% of the genes are assumed to be true negatives). GenoGAM (orange and blue) has a constantly higher recall with a lower false positive rate. **b** Replication of Fig. 3b from our previous study [3]. Area under the curve (AUC) for all possible quantile cutoffs from 0 to 1 in steps of 0.01. GenoGAM 1.0 (blue) and GenoGAM 2.0 (orange) are almost identical and are thus largely overlapping. Up to a cutoff of 0.6, GenoGAM (orange and blue) performs consistently better than all competitor methods by around 0.03-0.04 points above the second best method (csaw and DESeq2, pink and green, respectively). The entire range of quantile cutoffs is shown out of completeness, reasonable values are between 0.15 and 0.25. **c** Replication of supplementary figure S9C from our previous study [3]. Proportion of significant peaks within 30 bp of motif center and 95% bootstrap confidence interval (error bars) for six ENCODE transcription factors (CEBPB, CTCF, USF1, MAX, PAX5, YY1) on chromosome 22 and for the yeast TFIIB dataset

genome datasets on a typical lab server. Runtime for mega-base genomes like the yeast genome is within minutes on a standard PC. Finally, our algorithmic improvements apply to GAMs of long longitudinal data and can therefore be relevant for a broader community beyond the field of genomics.

Availability and requirements

Project name: GenoGAM
Project home page: https://github.com/gstricker/fastGenoGAM
Operating system(s): Platform independent
Programming language: R, C++
Other requirements: R 3.4.1 (https://cran.r-project.org/) or higher
License: GPL-2

Abbreviations

ChIP-Seq: chromatin immunoprecipitation with massively parallel DNA sequencing; GAM: Generalized additive model; HDF5: Hierarchical data format; IP: Immunoprecipitation; NB: Negative binomial distribution; P-splines: Penalized B-splines

Acknowledgements

We thank Alexander Bertram, Parham Solaimani, Martin Morgan and Hervé Pagès for support and advice during the implementation of the second version of GenoGAM, Thomas Huckle and Simon Wood for fruitful discussions and advice on sparse matrix methods.

Funding

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 633974 (J.G. and G.S.). This work was supported by the German Research Foundation (DFG) and the Technical University of Munich within the funding program Open Access Publishing. The funding bodies played no role in the design of the study, the collection, analysis, or interpretation of data or in writing the manuscript.

Availability of data and materials

The datasets analyzed during the current study are available from ENCODE:

- CEBPB: https://www.encodeproject.org/experiments/ENCSR000EHE
- FOXA1: https://www.encodeproject.org/experiments/ENCSR267DFA
- IRF9: https://www.encodeproject.org/experiments/ENCSR926KTP
- KLF1: https://www.encodeproject.org/experiments/ENCSR550HCT
- MAFG: https://www.encodeproject.org/experiments/ENCSR818DQV
- MNT: https://www.encodeproject.org/experiments/ENCSR261EDU
- NRF1: https://www.encodeproject.org/experiments/ENCSR135ANT

Yeast data analyzed during this study is included in this published article from Thornton et al. [23].

Authors' contributions
Conceived the project and supervised the work: JG Developed the software and carried out the analysis: GS, MG and JG Wrote the manuscript: GS and JG All authors read and approved the final version of the manuscript.

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Competing interests
The authors declare that they have no competing interests.

Publisher's Note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 9 February 2018 Accepted: 12 June 2018
Published online: 27 June 2018

References

1. Barski A, Cuddapah S, Cui K, Roh T-Y, Schones DE, Wang Z, Wei G, Chepelev I, Zhao K. High-Resolution Profiling of Histone Methylations in the Human Genome. Cell. 2007;129(4):823–37. https://doi.org/10.1016/j.cell.2007.05.009.
2. Johnson DS, Mortazavi A, Myers RM, Wold B. Genome-Wide Mapping of in Vivo Protein-DNA Interactions. Science. 2007;316(5830):1497.
3. Stricker G, Engelhardt A, Schulz D, Schmid M, Tresch A, Gagneur J. GenoGAM: Genome-wide generalized additive models for ChIP-Seq analysis. Bioinformatics. 2017. https://doi.org/10.1093/bioinformatics/btx150.
4. Hastie T, Tibshirani R. Generalized Additive Models. Stat Sci. 1986;1(3):297–318.
5. Huber W, Carey JV, Gentleman R, Anders S, Carlson M, Carvalho SB, Bravo CH, Davis S, Gatto L, Girke T, Gottardo R, Hahne F, Hansen DK, Irizarry AR, Lawrence M, Love IM, MacDonald J, Obenchain V, Ole's KA, Pag'es H, Reyes A, Shannon P, Smyth KG, Tenenbaum D, Waldron L, Morgan M. Orchestrating high-throughput genomic analysis with Bioconductor. Nat Methods. 2015;12(2):115–21.
6. Wood S. Generalized Additive Models: An Introduction with R. Boca Rota: Chapman and Hall/CRC; 2006.
7. Wood SN. Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. J R Stat Soc Ser B Stat Methodol. 2011. https://doi.org/10.1111/j.1467-9868.2010.00749.x.
8. De Boor C. A Practical Guide to Splines vol 27. New York: Springer; 1978.
9. Eilers PHC, Marx BD. Flexible smoothing with B -splines and penalties. Stat Sci. 1996;11(2):89–121. https://doi.org/10.1214/ss/1038425655.
10. Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biol. 2011;15. https://doi.org/10.1186/s13059-014-0550-8.
11. The HDF Group. Hierarchical Data Format Version 5. http://www.hdfgroup.org/HDF5. Accessed 02 June 2018.
12. Pagès H. HDF5Array: HDF5 back end for DelayedArray objects. 2017. https://bioconductor.org/packages/release/bioc/html/HDF5Array.html. Accessed 02 June 2018.
13. Fischer B, Pau G, Smith M. rhdf5: HDF5 interface to R. 2017. https://bioconductor.org/packages/release/bioc/html/rhdf5.html. Accessed 02 June 2018.
14. Bates D, Maechler M. Matrix: Sparse and Dense Matrix Classes and Methods. 2017. https://cran.r-project.org/package=Matrix. Accessed 02 June 2018.
15. Zammit-Mangion A. sparseinv: Computation of the Sparse Inverse Subset. 2018. https://cran.r-project.org/web/packages/sparseinv/index.html. Accessed 02 June 2018.
16. Davis TA. SuiteSparse: A suite of sparse matrix software. http://faculty.cse.tamu.edu/davis/suitesparse.html. Accessed 02 June 2018.
17. Li XS. An Overview of SuperLU: Algorithms, Implementation, and User Interface. ACM Transactions on Mathematical Software (TOMS) - Special issue on the Advanced CompuTational Software (ACTS) Collection. 2005;31(3):302–25.
18. Campbell YE, Davis TA. Computing the Sparse Inverse Subset : An inverse multifrontal approach. Technical report. Gainesville, FL: Computer and Information Sciences Deparment, University of Florida; 1995.
19. Davis TA. Direct Methods for Sparse Linear Systems. Philadelphia: Society for Industrial and Applied Mathematics; 2006. https://doi.org/10.1137/1.9780898718881.
20. Rue H, Held L. Gaussian Markov Random Fields: Theory and Applications. Boca Rota: Chapman and Hall/CRC; 2005.
21. Lee CYY, Wand MP. Streamlined mean field variational Bayes for longitudinal and multilevel data analysis. Biom J. 2016;58(4):868–95. https://doi.org/10.1002/bimj.201500007.
22. Wood SN. Comment. J Am Stat Assoc. 2017;112(517):164–6. https://doi.org/10.1080/01621459.2016.1270050.
23. Thornton JL, Westfield GH, Takahashi Y-H, Cook M, Gao X, Woodfin AR, Lee J-S, Morgan MA, Jackson J, Smith ER, Couture J-F, Skiniotis G, Shilatifard A. Context dependency of Set1/ COMPASS-mediated histone H3 Lys4 trimethylation. Genes Dev. 2014;28(2):115–20. https://doi.org/10.1101/gad.232215.113.

Ready to submit your research? Choose BMC and benefit from:

- fast, convenient online submission
- thorough peer review by experienced researchers in your field
- rapid publication on acceptance
- support for research data, including large and complex data types
- gold Open Access which fosters wider collaboration and increased citations
- maximum visibility for your research: over 100M website views per year

At BMC, research is always in progress.

Learn more biomedcentral.com/submissions