Faster model matrix crossproducts for large generalized linear models with discretized covariates

Zheyuan Li[2] · **Simon N. Wood**[1]

Abstract

Wood et al. (J Am Stat Assoc 112(519):1199–1210, 2017) developed methods for fitting penalized regression spline based generalized additive models, with of the order of 10^4 coefficients, to up to 10^8 data. The methods offered two to three orders of magnitude reduction in computational cost relative to the most efficient previous methods. Part of the gain resulted from the development of a set of methods for efficiently computing model matrix products when model covariates each take only a discrete set of values substantially smaller than the sample size [generalizing an idea first appearing in Lang et al. (Stat Comput 24(2):223–238, 2014)]. Covariates can always be rounded to achieve such discretization, and it should be noted that the covariate discretization is marginal. That is we do not rely on discretizing covariates jointly, which would typically require the use of very coarse discretization. The most expensive computation in model estimation is the formation of the matrix cross product $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$ where $\mathbf{X}$ is a model matrix and $\mathbf{W}$ a diagonal or tri-diagonal matrix. The purpose of this paper is to present a simple, novel and substantially more efficient approach to the computation of this cross product. The new method offers, for example, a 30 fold reduction in cross product computation time for the Black Smoke model dataset motivating Wood et al. (2017). Given this reduction in computational cost, the subsequent Cholesky decomposition of $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$ and follow on computation of $(\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X})^{-1}$ become a more significant part of the computational burden, and we also discuss the choice of methods for improving their speed.

Keywords Generalized additive model · Fast regression · BLAS

1 Introduction

A rate limiting step in computations involving large scale regression models is often the computation of weighted crossproducts, $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$, of the model matrix, $\mathbf{X}$, where $\mathbf{W}$ is diagonal (or in this paper sometimes tri-diagonal). When each covariate of the model results in several model matrix columns, as is the case in generalized additive models (GAM) or mixed models, then substantial efficiencies are possible. The key is to exploit the fact that irrespective of dataset size, n, most covariates only take one of a relatively modest number of discrete values, and even when that is not the case we can discretize each covariate to $\Theta(n^{1/2})$ rounded values without statistical loss. For example 10^7 daily temperature records are likely to contain only a few hundred distinct values, being

recorded only to the nearest tenth of a degree. Similarly data from a network of fixed monitoring stations contain only a fixed number of location co-ordinates, irrespective of data set size. This paper provides new algorithms for computing $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$ from discretized covariates, that are more efficient than previous algorithms, thereby substantially reducing the computational burden of estimating large GAMs of large data sets.

In its most basic form a GAM is a generalized linear model in which the linear predictor depends on unknown smooth functions, f_j, of covariates x_j (possibly vector valued). That is

$$g(\mu_i) = \mathbf{A}_i\boldsymbol{\theta} + \sum_{j=1}^{J} f_j(x_{ji}), \quad y_i \sim \mathrm{EF}(\mu_i, \phi) \tag{1}$$

where g is a known link function, $\mathbf{A}_i\boldsymbol{\theta}$ a parametric part of the linear predictor and EF some exponential family distribution with location μ_i and scale ϕ (Hastie and Tibshirani 1990). For practical estimation we use reduced rank spline basis expansions for the f_j, with basis size

✉ Simon N. Wood
simon.wood@bristol.ac.uk

1   School of Mathematics, University of Bristol, Bristol, UK

2   Mathematical Sciences, University of Bath, Bath, UK

typically $\Theta(n^{1/9})$ to $\Theta(n^{1/5})$ (see e.g. Wood 2017). In consequence the GAM becomes a richly parameterized generalized linear model with model matrix, $\mathbf{X}$, containing $\mathbf{A}$ and the evaluated spline basis functions. Inference with (1) also requires that we control the degree of smoothness of the f_j. This can be achieved by adding to the log likelihood a set of quadratic smoothing penalties on the spline basis coefficients, each weighted by a smoothing parameter, λ_j (e.g. Green and Silverman 1994). The smoothing parameters can be estimated by cross validation, for example. Alternatively the penalties can be induced by Gaussian priors on the spline basis coefficients (e.g. Silverman 1985), in which case inference about the λ_j can be Bayesian or empirical Bayesian, with the empirical Bayes approach being computationally efficient.

As mentioned above, the rate limiting computation in GAM inference is the computation of the matrix inner product $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$ where $\mathbf{X}$ is an $n \times p$ model matrix and $\mathbf{W}$ a diagonal or tri-diagonal weight matrix. Lang et al. (2014) recognised that if $\mathbf{X}$ depends on a single covariate which takes only $m \ll n$ discrete values then $\mathbf{X}$ has only m distinct rows, $\bar{\mathbf{X}}$, say. Given an index vector k, such that $\mathbf{X}(i,) = \bar{\mathbf{X}}(k(i),)$ then efficient computation can be based on $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X} = \bar{\mathbf{X}}^\mathsf{T}\bar{\mathbf{W}}\bar{\mathbf{X}}$ where $\bar{W}_{jj} = \sum_{k(i)=j} W_{ii}$ ($\mathbf{W}$ diagonal). Notice how discretization reduces the matrix product operations count from $O(np^2)$ to $O(mp^2) + O(n)$.

When the model matrix depends on multiple covariates then matters are somewhat more complicated. The obvious approach is to discretize the covariates jointly onto a grid and simply use the Lang et al. (2014) algorithm, but to maintain computational efficiency the grid then has to become coarser and coarser as the number of covariates increases, and the errors from discretisation rapidly exceed the statistical error. The alternative is to find ways to exploit discretization when covariates are discretized individually (marginally), and Wood et al. (2017) provide a set of algorithms to do this. These latter methods include the important case of model interaction terms. The columns of $\mathbf{X}$ relating to an interaction are given by a row-Kronecker product of a set of *marginal* model matrices for each marginal covariate of the interaction. These marginal covariates and their marginal model matrices are discretized separately.

This paper provides new algorithms that improve on Wood et al. (2017) in two ways. Firstly they have substantially reduced leading order computational cost whenever the product of the number of unique values of a pair of covariates is less than the sample size, and secondly they are matrix oriented, rather than vector oriented, and are hence able to make better use of BLAS level optimization. To emphasise the scale of the computational efficiency gains produced by the Wood et al. (2017) methods combined with the enhancements suggested here, Fig. 1 contours the computational time
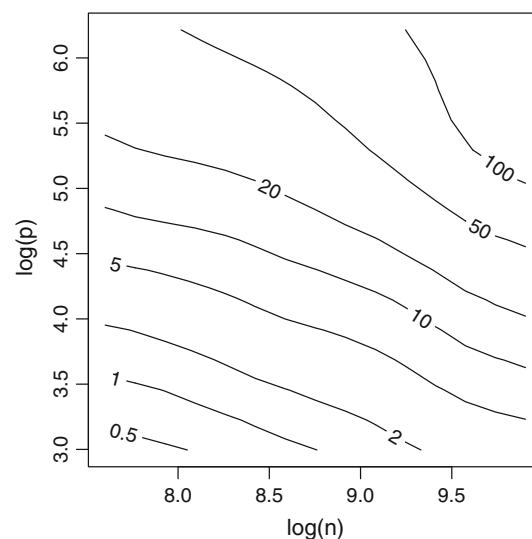

Fig. 1 Computation times for `gam` from R package `mgcv` divided by computation times using Wood et al. (2017) with the improved methods suggested here, against log sample sizes from 2000 to 20000 and log number of coefficients per smooth from 20 to 500. Computations are for a Gaussian additive model with 4 univariate smooths, fitted to data simulated using `mgcv` function `gamSim`. For comparability both methods use a single thread. To the lower left of the unit contour all model fits take less than 2 s. At the top right of the plot `gam` takes over 40 min

taken by the conventional algorithm implemented in the `gam` function of R package `mgcv`, divided by the time taken by the new methods, for some model-data combinations small enough to be fitted by `gam` in reasonable time. Section 5 provides practical illustration of the speed up provided by the new methods, using a big data example for which the `gam` methods would have a practically infeasible memory footprint, and where the theoretical speed up relative to the `gam` methods is many orders of magnitude.

2 The basic discrete cross product algorithms

The complete model matrix, $\mathbf{X}$, is made up column-wise of sub-matrices each relating to a single model term. Let $\mathbf{A}$ and $\mathbf{B}$ be two such sub-matrices, of dimension $n \times p_A$ and $n \times p_B$. The entire product $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$ is made up of blocks of the form $\mathbf{A}^\mathsf{T}\mathbf{W}\mathbf{B}$. For clarity of exposition, we initially assume that neither matrix represents an interaction term made up of row-Kronecker products, so that both are of the form $\mathbf{A}(i,) = \sum_{s=1}^{s_A} \bar{\mathbf{A}}(k_{As}(i),)$ where s_A is the number of index vectors used to define $\mathbf{A}$, and $\bar{\mathbf{A}}$ is $m_A \times p_A$. Definitions for $\mathbf{B}$ are similar. For standard generalized additive models or mixed models $s_{A/B} = 1$, but higher values are used to implement linear functionals of smooth terms, when these occur as model components, for example in

scalar on function regression[1] (see e.g. Ramsay and Silverman [2005](https://), Chap. 15). Further, let $\mathbf{w}$, $\mathbf{w}^+$ and $\mathbf{w}^-$ denote the leading, super and sub diagonals of tri-diagonal $\mathbf{W}$. Allowing tri-diagonal $\mathbf{W}$ accommodates simple AR1 correlation models (which have tridiagonal precision matrices), for example.

The basic idea is to work with the $m_A \times m_B$ matrix $\bar{\mathbf{W}}$ such that $\mathbf{A}^\mathsf{T}\mathbf{W}\mathbf{B} = \bar{\mathbf{A}}^\mathsf{T}\bar{\mathbf{W}}\bar{\mathbf{B}}$. Firstly, if $\mathbf{A} = \mathbf{B}$, $\mathbf{w}^+ = \mathbf{w}^- = \mathbf{0}$ and $s_A = 1$ then $\bar{\mathbf{W}}$ is diagonal and we can use the Lang et al. ([2014](https://)) algorithm. Let $\bar{\mathbf{w}}$, denote the diagonal of $\bar{\mathbf{W}}$.

Algorithm 0 (Lang et al. [2014](https://)).

1. Set all elements of $\bar{\mathbf{w}}$ to 0.
2. For $l = 1 \ldots n$ do $\bar{w}(k_{As}(l)) \mathrel{+}= w(l)$
3. Form $\bar{\mathbf{A}}^\mathsf{T}\bar{\mathbf{W}}\bar{\mathbf{A}}$.

This has cost $O(p_A^2 m_A) + O(n)$.

In more general circumstances $\bar{\mathbf{W}}$ will not be diagonal. In principle the algorithm for computing it is very simple, but there is an immediate problem. There is nothing to prevent $m_A m_B$ being much larger than n, so that $\bar{\mathbf{W}}$ requires excessive storage while having mostly zero entries (in the $s_A = s_B = 1$ diagonal $\mathbf{W}$ case there are at most n non-zero entries): to keep operations and memory costs sensibly bounded this has to be dealt with. Hence only if $n \geq m_A m_B$ will we use the following simple algorithm.

Algorithm 1 (*Weight accumulation*).

1. Set $m_A \times m_B$ matrix $\bar{\mathbf{W}}$ to 0.
2. For $s = 1 \ldots s_A$, $t = 1 \ldots s_B$, $l = 1 \ldots n$ do

   a. $\bar{W}(k_{As}(l), k_{Bt}(l)) \mathrel{+}= w(l)$
   b. If $l < n$ $\bar{W}(k_{As}(l), k_{Bt}(l+1)) \mathrel{+}= w^+(l)$
   c. If $l > 1$ $\bar{W}(k_{As}(l), k_{Bt}(l-1)) \mathrel{+}= w^-(l-1)$

3. Form $\bar{\mathbf{A}}^\mathsf{T}\bar{\mathbf{W}}\bar{\mathbf{B}}$ (use the multiplication ordering with lowest operation count).

Obviously steps 2b,c can be skipped if $\mathbf{W}$ is diagonal (the same will be true for Algorithms 2 and 3). The cost of this is the lower of $O(s_A s_B n) + O(p_A p_B m_A) + O(m_A m_B p_B)$ and $O(s_A s_B n) + O(p_A p_B m_B) + O(m_A m_B p_A)$.

In the $n < m_A m_B$ regime we need to deal with the intrinsic sparsity of $\bar{\mathbf{W}}$. The obvious option is to use sparse matrix methods to represent $\bar{\mathbf{W}}$. An algorithm implementing this approach is given in the "Appendix" and discussed there in detail. However, because handling of the sparse matrix

structures involves considerable non-locality of data, and therefore makes poor use of cache memory, the approach is often less efficient than the following methods that directly accumulate either $\mathbf{C} = \bar{\mathbf{W}}\bar{\mathbf{B}}$ or $\mathbf{D} = \bar{\mathbf{W}}^\mathsf{T}\bar{\mathbf{A}}$, depending on which has the lowest operations count. (In the case that one of $\bar{\mathbf{A}} = \mathbf{A}$ or $\bar{\mathbf{B}} = \mathbf{B}$ then the minimum memory option is chosen, obviously if $\bar{\mathbf{A}} = \mathbf{A}$ and $\bar{\mathbf{B}} = \mathbf{B}$ then we use a standard dense matrix inner product.)

Algorithm 2 (*Right accumulation*).

1. Set $m_A \times p_B$ matrix $\mathbf{C}$ to 0.
2. For $s = 1 \ldots s_A$, $t = 1 \ldots s_B$, $q = 1 \ldots p_B$, $l = 1 \ldots n$ do

   a. $C(k_{As}(l), q) \mathrel{+}= w(l)\bar{B}(k_{Bt}(l), q)$
   b. If $l < n$ $C(k_{As}(l), q) \mathrel{+}= w^+(l)\bar{B}(k_{Bt}(l+1), q)$
   c. If $l > 1$ $C(k_{As}(l), q) \mathrel{+}= w^-(l-1)\bar{B}(k_{Bt}(l-1), q)$

3. Form $\bar{\mathbf{A}}^\mathsf{T}\mathbf{C}$.

This has cost $O(s_B s_A n p_B) + O(p_A p_B m_A)$, i.e. essentially the same cost as Algorithm 1 if $m_A m_B = n$, assuming $s_A = s_B = 1$. Note that the l and q loops are ordered for optimal data-locality when matrices are stored in column major order (the convention in LAPACK and R, for example). The order should probably be reversed for row major order.

There is an alternative version of the algorithm that should be used if $\alpha s_A s_B n p_A + m_B p_A p_B < \alpha s_A s_B n p_B + m_A p_A p_B$, where α is the number of operations for steps a - c divided by 2.

Algorithm 3 (*Left accumulation*).

1. Set $m_B \times p_A$ matrix $\mathbf{D}$ to 0.
2. For $s = 1 \ldots s_A$, $t = 1 \ldots s_B$, $q = 1 \ldots p_A$, $l = 1 \ldots n$ do

   a. $D(k_{Bt}(l), q) \mathrel{+}= w(l)\bar{A}(k_{As}(l), q)$
   b. If $l < n$ $D(k_{Bt}(l), q) \mathrel{+}= w^-(l)\bar{A}(k_{As}(l+1), q)$
   c. If $l > 1$ $D(k_{Bt}(l), q) \mathrel{+}= w^+(l-1)\bar{A}(k_{As}(l-1), q)$

3. Form $\mathbf{D}^\mathsf{T}\bar{\mathbf{B}}$.

This has cost $O(s_B s_A n p_A) + O(p_A p_B m_B)$.

In principle using a sparse matrix representation of $\bar{\mathbf{W}}$, as in the "Appendix", reduces the cost of forming $\mathbf{D}$ to $O(n) + O(n_u p_A)$, for the $s_A = s_B = 1$ case, where n_u is the number of non-zeroes in $\bar{\mathbf{W}}$. Since $n_u \leq n$ this potentially represents a saving, provided that the overheads of using sparse matrices are low enough to not outweigh the n_u/n advantage. Similar arguments apply to $\mathbf{C}$. Of course since we have no guarantee that $n_u < n$, the *worst case* performance of the sparse approach will be the same as that of

---

[1] Weights can be accommodated in the summation by treating them as a one dimensional marginal in a tensor product interaction term, discussed in Sect. 3.

Algorithms 2 and 3 in leading order cost terms, but worse in practice because of the extra computational overhead. In our reported timings we use the sparse matrix approach in place of Algorithms 2 or 3 only when p_A or p_B is greater than 15 so that there is some real chance that the savings from sparsity outweigh the overheads (n_u can unfortunately not be obtained at lower cost than the full sparse accumulation algorithm).

In principle the equivalent Wood et al. (2017) algorithms have cost that is the lower of $O(s_B s_A n p_A) + O(p_A p_B m_B)$ and $O(s_B s_A n p_B) + O(p_A p_B m_A)$. Hence we only achieve a reduction in leading order cost when it is possible to use Algorithms 0 or 1, but then the savings can be large, for example a factor $O(n m_B^{-1} m_A^{-1})$ for Algorithm 1. However, when Algorithms 2 or 3 are used the leading order count does not tell the whole story. Firstly the constant of proportionality on the $O(s_B s_A n p_A)$ terms is higher for Wood et al. (2017), and secondly the Wood et al. (2017) methods are entirely vector based, and are therefore unable to make good use of optimized level 3 BLAS routines, unlike the methods proposed here. Hence the methods proposed here are always an improvement on Wood et al. (2017), and often a very substantial improvement.

2.1 Proof of algorithm correctness

Denote the desired cross product as $\mathbf{F} = \mathbf{A}^\mathsf{T} \mathbf{W} \mathbf{B}$, and assume that $\mathbf{W}$ is diagonal. From the definition of the storage convention we have

$$\mathbf{F} = \sum_i \bar{\mathbf{A}}(k_A(i),)^\mathsf{T} w_i \bar{\mathbf{B}}(k_B(i),)$$

(so each term in the summation is a rank one $p_A \times p_B$ matrix). Rows p and q of $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ co-occur in the summation whenever $k_A(i) = p$ and $k_B(i) = q$. Hence we can re-write the summation as

$$\mathbf{F} = \sum_{p,q} \bar{\mathbf{A}}(p,)^\mathsf{T} \bar{W}_{pq} \bar{\mathbf{B}}(q,) \quad \text{where} \quad \bar{W}_{pq} = \sum_{\substack{k_A(i)=p \\ k_B(i)=q}} w_i.$$

But $\bar{W}_{pq}$ is just element p, q of $\bar{\mathbf{W}}$ accumulated by Algorithm 1, and the preceding summation is simply $\mathbf{F} = \bar{\mathbf{A}}^\mathsf{T} \bar{\mathbf{W}} \bar{\mathbf{B}}$, confirming the correctness of Algorithm 1 in the diagonal case with $s_A = s_B = 1$.

Now consider the formation of $\mathbf{C} = \bar{\mathbf{W}} \bar{\mathbf{B}}$. We have

$$C(p, j) = \sum_q \bar{W}_{pq} \bar{B}(q, j) = \sum_q \sum_{\substack{k_A(i)=p \\ k_B(i)=q}} w_i \bar{B}(q, j),$$

and hence

$$C(k_A(i), j) = \sum_q \sum_{k_B(i)=q} w_i \bar{B}(q, j) = \sum_i w_i \bar{B}(k_B(i), j),$$

which is Algorithm 2. Algorithm 3 follows similarly.

Correctness of the algorithms in the tri-diagonal case follows by applying similar arguments to each diagonal and summing the results. When $s_A = s_B = 1$ does not hold, correctness follows immediately by linearity.

3 Discrete cross product algorithms for interaction terms

We now return to the case in which either or both of $\mathbf{A}$ and $\mathbf{B}$ are model matrix components relating to interaction terms, and are hence constructed as row-Kronecker products of a set of marginal model matrices (each relating to one of the interacting covariates). This includes the simple case in which a model term is multiplied by a known covariate, for example $f(x_i) z_i$, where x_i and z_i are both covariates: the multiplying covariate is treated as a single column marginal model matrix. For the moment suppose that there is only one index vector per marginal. We denote the marginal model matrices of $\mathbf{A}$, by $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{d_A}$, where each marginal model matrix has compact representation $\mathbf{A}_j(i,) = \bar{\mathbf{A}}_j(k_j^A(i),)$, and k_j^A is the index vector for the jth marginal of the interaction. Then $\mathbf{A} = \mathbf{A}_1 \odot \mathbf{A}_2 \odot \cdots \odot \mathbf{A}_{d_A}$, where '$\odot$' denotes the row-Kronecker product such that $\mathbf{A}(i,) = \mathbf{A}_1(i,) \otimes \mathbf{A}_2(i,) \otimes \cdots \otimes \mathbf{A}_{d_A}(i,)$. Also let $\dot{\mathbf{A}}$ denote the matrix such that $\mathbf{A} = \dot{\mathbf{A}} \odot \mathbf{A}_{d_A}$. In greater generality we might also be interested in $\mathbf{A}(i,) = \sum_{s=1}^{s_A} \bar{\mathbf{A}}_1(k_{1s}^A(i),) \otimes \bar{\mathbf{A}}_2(k_{2s}^A(i),) \otimes \cdots \otimes \bar{\mathbf{A}}_{d_A}(k_{d_A s}^A(i),)$, where there are s_A sets of indices to sum over. For the application of sum-to-zero constraints in this case we refer to the appendix of Wood et al. (2017). For maximum efficiency in the following, assume that the marginals are arranged so that $\mathbf{A}_{d_A}$ has the most columns. Similar definitions apply to $\mathbf{B}$.

Now let $\mathcal{D}(\mathbf{x})$ denote the diagonal matrix with $\mathbf{x}$ as its leading diagonal, let $\mathbf{C}_{\cdot, j}$ be column j of any matrix $\mathbf{C}$ and note that for a term with only one marginal $\dot{\mathbf{A}} = \mathbf{1}$ (similarly for $\mathbf{B}$). Then in the $s_A = s_B = 1$ case

$$\mathbf{A}^\mathsf{T} \mathbf{W} \mathbf{B} = \begin{pmatrix} \mathbf{A}_{d_a}^\mathsf{T} \mathcal{D}(\dot{\mathbf{A}}_{\cdot,1}) \mathbf{W} \mathcal{D}(\dot{\mathbf{B}}_{\cdot,1}) \mathbf{B}_{d_b} & \mathbf{A}_{d_a}^\mathsf{T} \mathcal{D}(\dot{\mathbf{A}}_{\cdot,1}) \mathbf{W} \mathcal{D}(\dot{\mathbf{B}}_{\cdot,2}) \mathbf{B}_{d_b} & \cdot \\ \mathbf{A}_{d_a}^\mathsf{T} \mathcal{D}(\dot{\mathbf{A}}_{\cdot,2}) \mathbf{W} \mathcal{D}(\dot{\mathbf{B}}_{\cdot,1}) \mathbf{B}_{d_b} & \mathbf{A}_{d_a}^\mathsf{T} \mathcal{D}(\dot{\mathbf{A}}_{\cdot,2}) \mathbf{W} \mathcal{D}(\dot{\mathbf{B}}_{\cdot,2}) \mathbf{B}_{d_b} & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

Each block, $\mathbf{A}_{d_a}^\mathsf{T} \mathcal{D}(\dot{\mathbf{A}}_{\cdot,i}) \mathbf{W} \mathcal{D}(\dot{\mathbf{B}}_{\cdot,j}) \mathbf{B}_{d_b}$, of this expression can be computed by the Algorithms 0–3 of the previous section (or the "Appendix"), upon replacement of the tridiagonal matrix $\mathbf{W}$ by tridiagonal matrix $\mathcal{D}(\dot{\mathbf{A}}_{\cdot,i}) \mathbf{W} \mathcal{D}(\dot{\mathbf{B}}_{\cdot,j})$. In the case in which s_A and s_B are not both 1, so we have to iterate over indices s and/or t, then $\mathbf{W}$ is replaced by $\mathcal{D}(\dot{\mathbf{A}}_{\cdot,i}^s) \mathbf{W} \mathcal{D}(\dot{\mathbf{B}}_{\cdot,j}^t)$, where the superscripts s and t allow for the change in index vectors and hence $\dot{\mathbf{A}}$ and $\dot{\mathbf{B}}$ with s and t.

4 Parallelization and other numerically costly operations

Since $\mathbf{X}^{\mathsf{T}}\mathbf{W}\mathbf{X}$ is made up of a number of $\mathbf{A}^{\mathsf{T}}\mathbf{W}\mathbf{B}$ blocks, it is very easy to parallelize the matrix cross product by computing different blocks in different threads, using openMP (OpenMP Architecture Review Board 2008). When there are tensor product terms present there is a choice to be made about whether to parallelize at the coarse 'whole term block' level, or at the finer level given by the sub-blocks resulting from the tensor product structure. Load balancing is typically slightly better with the finer block structure, and is in either case improved by processing blocks in order of decreasing computational cost.

The formation of $\mathbf{X}^{\mathsf{T}}\mathbf{W}\mathbf{X}$ is typically the most costly part of the Wood et al. (2017) estimation method, but the approach also requires the Cholesky decomposition of $\mathbf{X}^{\mathsf{T}}\mathbf{W}\mathbf{X} + \mathbf{S}_\lambda$ where $\mathbf{S}_\lambda$ is a positive semi-definite matrix determined by the smoothing penalties, plus the computation of $(\mathbf{X}^{\mathsf{T}}\mathbf{W}\mathbf{X} + \mathbf{S}_\lambda)^{-1}$ from the Cholesky factor. The original Wood et al. (2017) method implemented a parallel version of the block Cholesky method of Lucas (2004) followed by a parallel formation of $(\mathbf{X}^{\mathsf{T}}\mathbf{W}\mathbf{X} + \mathbf{S}_\lambda)^{-1}$: the implementations scaled well and had good performance relative to LAPACK's Cholesky routines based on the reference BLAS, but were poor compared to LAPACK using a tuned BLAS, such as OpenBLAS (Xianyi et al. 2014). These deficiencies can become rate limiting when the new $\mathbf{X}^{\mathsf{T}}\mathbf{W}\mathbf{X}$ methods are used, and we therefore re-implemented the methods to use level 3 BLAS routines wherever possible, while still parallelizing as appropriate via openMP. In this way it is possible to produce routines that give reasonable multi-core scaling for users of the reference BLAS, while also exploiting an optimized BLAS when this is used (albeit with less good scaling).

5 Example

To illustrate the practical improvements offered by the new algorithms, we revisit the daily black smoke pollution monitoring data and model motivating Wood et al. (2017). A key message is that on the same hardware (twin Intel E5-2670 v3 CPUs each with 12 physical cores) we are able to reduce the model estimation time from just over an hour for the original Wood et al. (2017) method to less than 5 min with the new crossproduct methods and improved BLAS use. We also achieve 7.5 min estimation time on a mid-range laptop (Intel i5-6300 with 2 physical cores).

The UK black smoke monitoring network operated for more than 4 decades from the early 1960s, and was set up after the UK clean air act which followed the severe London smog episodes of the 1950s. At any one time the network consisted of up to 1269 fixed stations distributed over 2862 distinct locations, although by the time of the network closure in 2005, only 73 stations remained. The stations recorded daily particulate pollution levels (black smoke) in units of $\mu\mathrm{g}\ \mathrm{m}^{-3}$. Smooth additive modelling of black smoke measurements is desirable for individual short term exposure estimation for epidemiological purposes, and to partially alleviate the effects of the network design being non-random (with more stations in high pollution areas than in low, and higher probability of removing stations in low pollution areas).

The model structure used in Wood et al. (2017) and here is,

$$
\begin{aligned}
\log(\mathrm{bs}_i) = {} & f_1(\mathrm{y}_i) + f_2(\mathrm{doy}_i) + f_3(\mathrm{dow}_i) + f_4(\mathrm{y}_i, \mathrm{doy}_i) \\
& + f_5(\mathrm{y}_i, \mathrm{dow}_i) + f_6(\mathrm{doy}_i, \mathrm{dow}_i) \\
& + f_7(\mathrm{n}_i, \mathrm{e}_i) + f_8(\mathrm{n}_i, \mathrm{e}_i, \mathrm{y}_i) + f_9(\mathrm{n}_i, \mathrm{e}_i, \mathrm{doy}_i) \\
& + f_{10}(\mathrm{n}_i, \mathrm{e}_i, \mathrm{dow}_i) + f_{11}(\mathrm{h}_i) + f_{12}(\mathrm{T}_i^0, \mathrm{T}_i^1) \\
& + f_{13}(\bar{\mathrm{T}1}_i, \bar{\mathrm{T}2}_i) + f_{14}(\mathrm{r}_i) + \alpha_{k(i)} + b_{\mathrm{id}(i)} + e_i
\end{aligned}
\tag{2}
$$

where y, doy and dow denote, year, day of year and day of week; n and e denote location as kilometres north and east; h and r are height (elevation of station) and cube root transformed rainfall (unfortunately only available as monthly average); T^0 and T^1 are daily minimum and maximum temperature, while $\bar{\mathrm{T}1}$ and $\bar{\mathrm{T}2}$ are daily mean temperature on and two days previously; $\alpha_{k(i)}$ is a fixed effect for the site type k of the ith observation (type is one of R (rural), A (industrial), B (residential), C, (commercial), D (city/town centre), X (mixed) or M (missing)); $b_{\mathrm{id}(i)}$ is a random effect for the idth station, while e_i is a Gaussian error term following an AR1 process at each site (the AR parameter being obtained by profile likelihood). Given basis expansions for all the terms the model has 7320 coefficients and there are 9451232 observations.

Table 1 gives timings in minutes for a single model estimation, excluding the model set up time (initial data handling and basis set up) which takes 3.2 min. Timings are given for computing with 1, 4 and 8 cores using single threaded reference BLAS and OpenBLAS, and simple OpenMP parallelization of the cross-product, Cholesky decomposition and subsequent inversion. We also report a hybrid approach in which we used a multi-threaded OpenBLAS and OpenMP parallelization for the cross product in our code, set up to ensure that there were never more threads than cores (using 24 cores for the multi-threaded BLAS is slower than single threaded for this example). The original Wood et al. (2017) method took about 1 h for the same example using 10 cores on the same hardware, and around 6.5 h with a single core. So the improvement with the new methods is substantial.

In addition we compared the timings of the new method and the Wood et al. (2017) method for the crossproduct alone,

Table 1 Timings in minutes for black smoke model estimation using the new methods, for various combinations of BLAS and multi-threading

BLAS\Threads	1	4	8
Reference	126.3	43.4	30.7
OpenBLAS	12.3	6.9	5.8
\Threads	6/1	6/4	4/6
OpenBLAS MT	9.6	5.0	4.8

The reference and OpenBLAS are single threaded, so that the parallelism is within our implementation. OpenBLAS MT is multi-threaded and the two figures for threading give the number of threads used by the BLAS and then the number of threads used within our code. The total number of threads used is the product of the two numbers, since each thread of our code calls the BLAS: hence the 6/4 and 4/6 columns both use 24 threads in total. The original Wood et al. (2017) method took an hour using 10 threads, and 6.5 h using one thread on the same hardware

Table 2 Timings in seconds for the formation of $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$ using the new methods and the Wood et al. (2017) methods with different numbers of threads, and different sized sub-samples

n	10^5		10^6		All	
Threads	New	Old	New	Old	New	Old
1	35	78	125	404	230	7024
4	13	23	45	133	65	1807
8	13	13	34	97	39	1086

The equal timing for 8 threads at $n = 10^5$ reflects slightly unfortunate load balancing for this example with the new method. The improvements of the new method for $n = 10^6$ are more reliant on the sparse matrix accumulation method than the other results. Without the sparse methods the $n = 10^5$ and all data timings barely change, while the $n = 10^6$ timings are around 2.5 times larger, offering only modest improvement on the old methods

using a single threaded OpenBLAS and simple OpenMP parallelization. The results are shown in Table 2. We compared timings for the full dataset of 9451232 observations and two random subsamples of size 10^5 and 10^6. For the full dataset the natural discretizations of the data are such that all the term to term crossproducts use Algorithm 1. For the subsamples, respectively 40% and 20% of these cross-products use Algorithms 2, 3 or 4, but these are the crossproducts involving the majority of the work.

As well as illustrating the substantial gains that can accrue from using the new methods, the results illustrate an interesting feature of the algorithms. Namely, although Algorithms 1–3 have essentially the same leading order cost when $m_A m_b = n$, in fact Algorithm 1 will be much faster when an optimized BLAS is used, because most of its cost is in level 3 BLAS operations. Hence, in principle, if we were willing to tolerate the extra storage costs, it would be worth using Algorithm 1 whenever $m_A m_b < kn$, where $k > 1$ is some constant related to the BLAS performance improvement.

6 Conclusions

The new methods presented here offer substantial reductions in the computational burden associated with estimating large generalized additive models (Gaussian latent process models) for large data sets. The discrete matrix crossproduct methods offer advantages for any regression model in which each covariate results in a term with several associated model matrix columns: models containing several factor variables are an obvious example beyond smooth regression models. Extension of the algorithms to banded $\mathbf{W}$ matrices with more than 3 bands is obvious, but we have not yet implemented this extension. Of course the methods are not useful in all settings. For example, when models have very large numbers of coefficients (larger than 10^4, or a small multiple of this) but a sparse model matrix, direct use of sparse matrix methods (e.g. Davis 2006) may be more appropriate. Note, however, that quite high levels of sparsity may be needed to ensure feasibility of sparse methods: for example, the model matrix for (2) would need to be about 99.5% zeroes before it required less than 10Gb of storage, or 99.95% if we wanted 10 times as many coefficients. Furthermore, depending on the model details, $\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}$ can be substantially less sparse than $\mathbf{X}$, or nearly dense in the worst cases.

The algorithms developed here are available in R package mgcv, from version 1.8-25.

Appendix: sparse accumulation of $\bar{\mathbf{W}}$

$\bar{\mathbf{W}}$ can be accumulated as a sparse matrix, to avoid excessive storage requirements, and to exploit the fact that it can have no more than n non-zero elements ($3n$ for tridiagonal $\mathbf{W}$). The resulting algorithm has lower operations cost than Algorithms 2 and 3, but there is unavoidable loss of data locality associated with handling the sparse matrices, so that practical performance is reduced by inefficient use of cache memory. The sparsity pattern of $\bar{\mathbf{W}}$ is not known in advance. To obtain it in advance we could resort to sorting each pair of k_a, k_b indices, but this involves more work than necessary, since we do not require the matrix elements to be ordered. We therefore adopt a hash table based solution.

In particular we use a hash function (as described in Press et al. 2007, Sect. 7.6.2) to convert row column index pairs to locations (between 1 and n) in a hash table. Denote this

function by hash(i, j). A length n array, **sm**, then contains pointers to linked lists of matrix elements, with NULL denoting no entries. The elements, e, contain entries $e.i$, $e.j$, $e.w$ and e.next, respectively the row, column, value and pointer to the next linked list element. The latter is NULL at the end of a linked list. An array of elements is created before the algorithm is run, to provide a stack of elements to add to the link lists as needed. n elements are needed in the diagonal **W** case, and $3n$ in the tri-diagonal case, although these are both upper bounds on actual usage. The purpose of the hash function is to ensure that matrix entries are stored evenly over the elements of **sm** and hence that the linked lists starting at each element of **sm** each have at most a few elements.

Algorithm 4 (*Sparse accumulation*).

1. Set T to 1 or 3 depending on whether **W** is diagonal or tri-diagonal.
2. For $l = 1, \ldots, n$, for $t = 1 \ldots T$ repeat 3 to 5.
3. Dependent on t execute one of the following.

   $t = 1$: $i = k_a(l)$, $j = k_b(l)$, $\omega = w(l)$.
   $t = 2$: $i = k_a(l)$, $j = k_b(l + 1)$, $\omega = w^+(l)$.
   $t = 3$: $i = k_a(l + 1)$, $j = k_b(l)$, $\omega = w^-(l)$. If $l = n - 1$ set $T = 1$.

4. $k = \text{hash}(i, j)$
5. Set e to the record pointed to by **sm**$[k]$:

   (a) While e is non-NULL.
       - If $e.i = i$ and $e.j = j$ then set $e.w \mathrel{+}= \omega$ and go to the next t or l.
       - Otherwise set e to the record pointed to by e.next.
   (b) No match, so insert a new record in the linked list at **sm**$[k]$ with $e.i = i$, $e.j = j$ and $e.w = \omega$.

The linked lists pointed to by **sm** now contain the unique elements of $\bar{\mathbf{W}}$. In principle we can use this structure directly with the diagonal **W** versions of Algorithms 2 or 3 to obtain **C** or **D**, however much better data locality and hence cache efficiency results from reading the i, j and w records out to arrays k'_a, k'_b and w' before computing **C** or **D**.

Notice that the accumulation of $\bar{\mathbf{W}}$ has only $O(n)$ cost, but involves considerable non-localised memory access. The matrix multiplications to form **C** and **D** have lower cost than Algorithms 2 and 3 whenever the number of unique matrix elements in $\bar{\mathbf{W}}$ is less than n. The data non-locality overheads mean that in reality we have to choose when to use the sparse methods. In the work reported here we use Algorithms 2 and 3 when p_A or p_B are less than 15, and the sparse method otherwise (obviously we use Algorithms 0 and 1 wherever possible). The choice of 15 is somewhat arbitrary, and we did not seek to tune it. All that really matters is that it is not too far from the data retrieval latency (i.e. the time it takes to retrieve a data item from main (not cache) memory divided by the length of a CPU cycle).

References

Davis, T.A.: Direct Methods for Sparse Linear Systems. SIAM, Philadelphia (2006)

Green, P.J., Silverman, B.W.: Nonparametric Regression and Generalized Linear Models. Chapman & Hall, London (1994)

Hastie, T., Tibshirani, R.: Generalized Additive Models. Chapman & Hall, London (1990)

Lang, S., Umlauf, N., Wechselberger, P., Harttgen, K., Kneib, T.: Multilevel structured additive regression. Stat. Comput. **24**(2), 223–238 (2014)

Lucas, C.: LAPACK-style codes for level 2 and 3 pivoted Cholesky factorizations. In: LAPACK Working Paper (2004)

OpenMP Architecture Review Board (2008) OpenMP application program interface version 3.0

Press, W., Teukolsky, S., Vetterling, W., Flannery, B.: Numerical Recipes, 3rd edn. Cambridge University Press, Cambridge (2007)

Ramsay, J., Silverman, B.: Functional Data Analysis, 2nd edn. Springer, Berlin (2005)

Silverman, B.W.: Some aspects of the spline smoothing approach to non-parametric regression curve fitting. J. R. Stat. Soc. Ser. B **47**(1), 1–53 (1985)

Wood, S.N.: Generalized Additive Models: An Introduction with R, 2nd edn. CRC Press, Boca Raton (2017)

Wood, S.N., Li, Z., Shaddick, G., Augustin, N.H.: Generalized additive models for gigadata: modelling the UK black smoke network daily data. J. Am. Stat. Assoc. **112**(519), 1199–1210 (2017)

Xianyi, Z., Qian, W., Chothia, Z.: OpenBLAS. http://xianyi.github.io/OpenBLAS (2014)