Taylor & Francis
Taylor & Francis Group

Check for updates

Penalized Estimation in Large-Scale Generalized Linear Array Models

Adam Lund, Martin Vincent, and Niels Richard Hansen

Department of Mathematical Sciences, University of Copenhagen, Copenhagen, Denmark

ABSTRACT

Large-scale generalized linear array models (GLAMs) can be challenging to fit. Computation and storage of its tensor product design matrix can be impossible due to time and memory constraints, and previously considered design matrix free algorithms do not scale well with the dimension of the parameter vector. A new design matrix free algorithm is proposed for computing the penalized maximum likelihood estimate for GLAMs, which, in particular, handles nondifferentiable penalty functions. The proposed algorithm is implemented and available via the R package `glamlasso`. It combines several ideas—previously considered separately—to obtain sparse estimates while at the same time efficiently exploiting the GLAM structure. In this article, the convergence of the algorithm is treated and the performance of its implementation is investigated and compared to that of `glmnet` on simulated as well as real data. It is shown that the computation time for `glamlasso` scales favorably with the size of the problem when compared to `glmnet`. Supplementary materials, in the form of R code, data, and visualizations of results, are available online.

1. Introduction

The generalized linear array models (GLAMs) were introduced in Currie, Durban, and Eilers (2006) as generalized linear models (GLMs) where the observations can be organized in an array and the design matrix has a tensor product structure. One main application treated in Currie, Durban, and Eilers (2006)—that will also be central to this article—is multivariate smoothing where data are observed on a multidimensional grid.

In this article, we present results on three-dimensional smoothing for two quite different real datasets where the aim was to extract a smooth mean signal. The first dataset contains voltage sensitive dye recordings of spiking neurons in a live ferret brain and was modeled in a Gaussian GLAM framework. The second dataset contains all registered Medallion taxi pickups in New York City during 2013 and was modeled in a Poisson GLAM framework. In both examples, we fitted an ℓ_1-penalized B-spline basis expansion to obtain a clear signal. For the taxi data, we also demonstrate how the ℓ_1-penalized fit lead to a lower error, compared to the nonpenalized fit, when trying to predict missing observations. Other potential applications include factorial designs and contingency tables.

Currie, Durban, and Eilers (2006) showed how the structure of GLAMs can be exploited for computing the maximum likelihood estimate and other quantities of importance for statistical inference. The penalized maximum likelihood estimate for a quadratic penalty function can also be computed easily by similar methods. The computations are simple to implement efficiently in any high-level language like R or MATLAB that supports fast numerical linear algebra routines. They exploit the GLAM structure to carry out linear algebra operations involving only the tensor factors—called array arithmetic,

see also De Boor (1979) and Buis and Dyksen (1996)—and they avoid forming the design matrix. This design matrix free approach offers benefits in terms of memory as well as time usage compared to standard GLM computations.

The approach of Currie, Durban, and Eilers (2006) has some limitations when the dimension p of the parameter vector becomes large. The $p \times p$ weighted cross-product of the design matrix has to be computed, and though this computation can benefit from the GLAM structure, a linear equation in the parameter vector remains to be solved. The computations can become prohibitive for large p. Moreover, the approach does not readily generalize to nonquadratic penalty functions like the ℓ_1-penalty or for that matter nonconvex penalty functions like the smoothly clipped absolute deviation (SCAD) penalty.

In this article, we investigate the computation of the penalized maximum likelihood estimate in GLAMs for a general convex penalty function. However, we note that by employing the multi-step adaptive lasso (MSA-lasso) algorithm from sec. 2.8.5 and 2.8.6 in Bühlmann and van de Geer (2011) our algorithm can easily be extended to handle nonconvex penalty functions. This modification is already implemented in the R package `glamlasso` for the SCAD-penalty, see Lund (2016). The convergence results presented in this article are, however, only valid for a convex penalty.

Algorithms considered in the literature hitherto for ℓ_1-penalized estimation in GLMs, see, for example, Friedman et al. (2010), cannot easily benefit from the GLAM structure, and typically they need the design matrix explicitly or at least direct access to its columns. Our proposed algorithm based on proximal operators is design matrix free—in the sense that the tensor product design matrix need not be computed—and can exploit

the GLAM structure, which results in an algorithm that is both memory and time efficient.

The article is organized as follows. In Section 2, GLAMs are introduced. In Section 3, our proposed GD-PG algorithm for computing the penalized maximum likelihood estimate is described. Section 4 presents two multivariate smoothing examples where the algorithm is used to fit GLAMs. This section includes a benchmark comparison between our implementation of the GD-PG algorithm in the R package glamlasso and the algorithm implemented in glmnet. Section 5 presents a convergence analysis of the proposed algorithm. In Section 6, a number of details on how the algorithm is implemented in glamlasso are collected. This includes details on how the GLAM structure is exploited, and the section also presents further benchmark results. Section 7 concludes the article with a discussion. Some technical and auxiliary definitions and results are presented in two appendices.

2. Generalized Linear Array Models

A generalized linear model (GLM) is a regression model of n independent real valued random variables $\mathcal{Y}_1, \ldots, \mathcal{Y}_n$, see Nelder and Wedderburn (1972). A generalized linear array model (GLAM) is a GLM with some additional structure of the data. We first introduce GLMs and then the special data structure for GLAMs.

With X an $n \times p$ design matrix, the linear predictor $\eta : \mathbb{R}^p \to \mathbb{R}^n$ is defined as

$$\eta(\theta) := X\theta \tag{1}$$

for $\theta \in \mathbb{R}^p$. With $g : \mathbb{R} \to \mathbb{R}$ denoting the link function, the mean value of $\mathcal{Y}_i$ is given in terms of $\eta_i(\theta)$ via the equation

$$g(\mathrm{E}(\mathcal{Y}_i)) = \eta_i(\theta). \tag{2}$$

The link function g is throughout assumed invertible with a continuously differentiable inverse.

The distribution of $\mathcal{Y}_i$ is, furthermore, assumed to belong to an exponential family, see Appendix B, which implies that the log-likelihood, $\theta \mapsto l(\eta(\theta))$, is given in terms of the linear predictor. With $y = (y_1, \ldots, y_n)^\top \in \mathbb{R}^n$ denoting a vector of realized observations of the variables $\mathcal{Y}_i$, the log-likelihood (with weights $a_i \geq 0$ for $i = 1, \ldots, n$) and its gradient are given as

$$l(\eta(\theta)) = \sum_{i=1}^n a_i(y_i \vartheta(\eta_i(\theta)) - b(\vartheta(\eta_i(\theta)))) \quad \text{and} \quad \tag{3}$$

$$\nabla_\theta l(\eta(\theta)) = X^\top u(\eta(\theta)), \tag{4}$$

respectively, where $\vartheta : \mathbb{R} \to \mathbb{R}$ denotes the canonical parameter function, and $u(\eta) := \nabla_\eta l(\eta)$ is the score statistic, see Appendix B.

The main problem considered in this article is the computation of the penalized maximum likelihood estimate (PMLE),

$$\theta^* := \operatorname*{arg\,min}_{\theta \in \mathbb{R}^p} -l(\eta(\theta)) + \lambda J(\theta), \tag{5}$$

where $J : \mathbb{R}^p \to (-\infty, \infty]$ is a proper, convex and closed penalty function, and $\lambda \geq 0$ is a regularization parameter controlling the amount of penalization. Note that J is allowed to take the value ∞, which can be used to enforce convex parameter

constraints. The objective function of this minimization problem is thus the penalized negative log-likelihood, denoted

$$F := -l + \lambda J, \tag{6}$$

where $-l$ is continuously differentiable.

For a GLAM, the vector y is assumed given as $y = \mathrm{vec}(Y)$ (the vec operator is discussed in Appendix A), where Y is an $n_1 \times \cdots \times n_d$ d-dimensional array. The design matrix X is assumed to be a concatenation of c matrices

$$X = [X_1 | X_2 | \ldots | X_c],$$

where the rth component is a tensor product,

$$X_r = X_{r,d} \otimes X_{r,d-1} \otimes \cdots \otimes X_{r,1}, \tag{7}$$

of d matrices. The matrix $X_{r,j}$ is an $n_j \times p_{r,j}$ matrix, such that

$$n = \prod_{j=1}^d n_j, \quad p_r := \prod_{j=1}^d p_{r,j}, \quad p = \sum_{r=1}^c p_r.$$

We let $\langle X_{r,j} \rangle := \langle X_{1,1}, \ldots, X_{c,d} \rangle$ denote the tuple of marginal design matrices.

The assumed data structure induces a corresponding structure on the parameter vector, θ, as a concatenation of c vectors,

$$\theta^\top = (\mathrm{vec}(\Theta_1)^\top, \ldots, \mathrm{vec}(\Theta_c)^\top),$$

with Θ_r a $p_{r,1} \times \cdots \times p_{r,d}$ d-dimensional array. We let $\langle \Theta_r \rangle := \langle \Theta_1, \ldots, \Theta_c \rangle$ denote the tuple of parameter arrays.

Given this structure it is possible to define a map, ρ, such that with $\theta_r = \mathrm{vec}(\Theta_r)$,

$$X_r \theta_r = \mathrm{vec}\left(\rho(X_{r,d}, \ldots, \rho(X_{r,2}, (\rho(X_{r,1}, \Theta_r))) \ldots)\right) \tag{8}$$

for $r = 1, \ldots, c$. The algebraic details of ρ are spelled out in Appendix A.

As a consequence of the array structure, the linear predictor can be computed using ρ without explicitly constructing X. The most obvious benefit is that no large tensor product matrix needs to be computed and stored. In addition, the array structure can be beneficial in terms of time complexity. As noted in Buis and Dyksen (1996), with $X_{r,j}$ being a square $n_r \times n_r$ matrix, say, the computation of the direct matrix-vector product in (8) has $O(n_r^{2d})$ time complexity, while the corresponding array computation has $O(dn_r^{d+1})$ time complexity. This reduced time complexity for $d \geq 2$ translates, as mentioned in the introduction, directly into a computational advantage for computing the PMLE with a quadratic penalty function, see Currie, Durban, and Eilers (2006). For nonquadratic penalty functions, the translation is less obvious, but we present one algorithm that is capable of benefitting from the array structure.

3. Penalized Estimation in a GLAM

In most situations, the PMLE must be computed by an iterative algorithm. We present an algorithm that solves the optimization problem (5) by iteratively optimizing a partial quadratic approximation to the objective function while exploiting the array structure. The proposed algorithm is a combination of a

gradient-based descent (GD) algorithm with a proximal gradient (PG) algorithm. The resulting algorithm, which we call GD-PG, thus consists of the following two parts:

- an outer GLAM enhanced GD loop
- an inner GLAM enhanced PG loop.

We present these two loops in the sections below postponing the details on how the array structure can be exploited to Section 6, where it is explained in detail how the two loops can be enhanced for GLAMs.

3.1. The Outer GD Loop

The outer loop consists of a sequence of descent steps based on a partial quadratic approximation of the objective function. This results in a sequence of estimates, each of which is defined in terms of a penalized weighted least-square estimate and whose computation involves an iterative choice of weights. The weights can be chosen so that the inner loop can better exploit the array structure.

For $k \in \mathbb{N}$ and $\theta^{(k)} \in \mathbb{R}^p$, let $\eta^{(k)} = \eta(\theta^{(k)})$ and $u^{(k)} = \nabla_\eta l(\eta^{(k)})$, let $W^{(k)}$ denote a positive definite diagonal $n \times n$ weight matrix, and let $z^{(k)}$ denote the n-dimensional vector (the working response) given by

$$z^{(k)} := (W^{(k)})^{-1}u^{(k)} + \eta^{(k)}. \quad (9)$$

The sequence $(\theta^{(k)})$ is defined recursively from an initial $\theta^{(0)}$ as follows. Given $\theta^{(k)}$ let

$$\tilde{\theta}^{(k+1)} := \underset{\theta \in \mathbb{R}^p}{\arg\min} \frac{1}{2n}\|\sqrt{W^{(k)}}(X\theta - z^{(k)})\|_2^2 + \lambda J(\theta) \quad (10)$$

denote the penalized weighted least-square estimate and define

$$\theta^{(k+1)} := \theta^{(k)} + \alpha_k(\tilde{\theta}^{(k+1)} - \theta^{(k)}), \quad (11)$$

where the stepsize $\alpha_k > 0$ is determined to ensure sufficient descent of the objective function, for example, by using the Armijo rule. A detailed convergence analysis is given in Section 5, where the relation to the class of gradient-based descent algorithms in Tseng and Yun (2009) is established.

3.2. The Inner PG Loop

The inner loop solves (10) by a proximal gradient algorithm. To formulate the algorithm, consider a generic version of (10) given by

$$x^* := \underset{x \in \mathbb{R}^p}{\arg\min}\, h(x) + \lambda J(x), \quad (12)$$

where $h : \mathbb{R}^p \to \mathbb{R}$ is convex and continuously differentiable. It is assumed that there exists a minimizer x^*. Define for $\gamma > 0$ the proximal operator, $\text{prox}_\gamma : \mathbb{R}^p \to \mathbb{R}^p$, by

$$\text{prox}_\gamma(z) = \underset{x \in \mathbb{R}^p}{\arg\min}\left\{\frac{1}{2}\|x - z\|_2^2 + \gamma J(x)\right\}.$$

The proximal operator is particularly easy to compute for a separable penalty function like the 1-norm or the squared 2-norm. Given a stepsize $\delta_k > 0$, initial values $x^{(0)} = x^{(1)} \in \mathbb{R}^p$ and an extrapolation sequence (ω_l) with $\omega_l \in [0, 1)$ define the

sequence $(x^{(l)})$ recursively by

$$y := x^{(l)} + \omega_l\left(x^{(l)} - x^{(l-1)}\right) \quad \text{and} \quad (13)$$

$$x^{(l+1)} := \text{prox}_{\delta_k\lambda}(y - \delta_k\nabla h(y)). \quad (14)$$

The choice of $\omega_l = 0$ for all $l \in \mathbb{N}$ gives the classical proximal gradient algorithm, see Parikh and Boyd (2014). Other choices of the extrapolation sequence, for example, $\omega_l = (l-1)/(l+2)$, can accelerate the convergence. Convergence results can be established if ∇h is Lipschitz continuous and δ_k is chosen sufficiently small—see Section 5 for further details.

For the convex function

$$h(\theta) := \frac{1}{2n}\|\sqrt{W^{(k)}}(X\theta - z^{(k)})\|_2^2, \quad (15)$$

we have that

$$\nabla h(\theta) = \frac{1}{n}X^\top W^{(k)}(X\theta - z^{(k)}). \quad (16)$$

This shows that $\nabla h(\theta)$ is Lipschitz continuous, and its explicit form in (16) indicates how the array structure can be exploited—see also Section 6.

3.3. The GD-PG Algorithm

The combined GD-PG algorithm is outlined as Algorithm 1. It is formulated using array versions of the model components. Especially, $U^{(k)}$ and $Z^{(k)}$ denote $n_1 \times \cdots \times n_d$ array versions of the score statistic, $u^{(k)}$, and the working response, $z^{(k)}$, respectively. Also $V^{(k)}$ is an $n_1 \times \cdots \times n_d$ array containing the diagonal of the $n \times n$ weight matrix $W^{(k)}$. The details on how the steps in the algorithm can exploit the array structure are given in Section 6.

Algorithm 1 GD-PG

Require: $\langle \Theta_r^{(0)}\rangle, \langle X_{r,j}\rangle$
1: **for** $k = 0$ to $K \in \mathbb{N}$ **do**
2:     given $\langle \Theta_r^{(k)}\rangle$: compute $U^{(k)}$, specify $V^{(k)}$ and compute $Z^{(k)}$
3:     specify the proximal stepsize δ_k
4:     given $\langle \Theta_r^{(k)}\rangle, V^{(k)}, Z^{(k)}, \delta_k$: compute $\langle \tilde{\Theta}_r^{(k+1)}\rangle$ by the inner PG loop
5:     given $\langle \Theta_r^{(k)}\rangle, \langle \tilde{\Theta}_r^{(k+1)}\rangle$: use a line search to compute $\langle \Theta_r^{(k+1)}\rangle$
6:     **if** convergence criterion is satisfied **then**
7:        break
8:     **end if**
9: **end for**

The outline of Algorithm 1 leaves out some details that are required for an implementation. In Step 2, the weights must be specified. In Section 5, we present results on convergence of the outer loop, which put some restrictions on the choice of weights. In Step 3, the proximal gradient stepsize must be specified. In Section 5, we give a computable upper bound on the stepsize that ensures convergence of the inner PG loop. Convergence with the same convergence rate can also be ensured for larger stepsizes if a backtracking step is added to the inner PG loop. In Step 4, $\langle \Theta_r^{(k)}\rangle$ is a natural choice of initial value in the inner PG loop, but this choice is not necessary to ensure convergence.

In Step 4 it is, in addition, necessary to specify the extrapolation sequence. Finally, in Step 5 a line search is required. In Section 5, convergence of the outer loop is treated when the Armijo rule is used.

4. Applications to Multidimensional Smoothing

As a main application of the GD-PG algorithm, we consider multidimensional smoothing, which can be formulated in the framework of GLAMs by using a basis expansion with tensor product basis functions. We present the framework below and report the results obtained for two real datasets.

4.1. A Generalized Linear Array Model for Smoothing

Letting $\mathcal{X}_1, \ldots, \mathcal{X}_d \subseteq \mathbb{R}$ denote d finite sets define the d-dimensional grid

$$\mathcal{G}_d := \mathcal{X}_1 \times \cdots \times \mathcal{X}_d.$$

The set $\mathcal{X}_j$ is the set of (marginal) grid points in the jth dimension and $n_j := |\mathcal{X}_j|$ denotes the number of such marginal points in the jth dimension. We have a total of $n := \prod_{j=1}^d n_j$ d-dimensional joint grid points, or d-tuples,

$$(x_1, \ldots, x_d) \in \mathcal{G}_d.$$

For each of the n grid points, we observe a corresponding grid value $y_{x_1, \ldots, x_d} \in \mathbb{R}$ assumed to be a realization of a real valued random variable $\mathcal{Y}_{x_1, \ldots, x_d}$ with finite mean. That is, the observations can be regarded as a d-dimensional array Y. With $g : \mathbb{R} \to \mathbb{R}$ a link function let

$$f(x_1, \ldots, x_d) := g(\mathrm{E}(\mathcal{Y}_{x_1, \ldots, x_d})), \quad (x_1, \ldots, x_d) \in \mathcal{G}_d. \quad (17)$$

The objective is to estimate f, which is assumed to possess some form of regularity as a function of $(x_1, \ldots, x_d)$. Assuming that f belongs to the span of p basis functions, $\phi_1, \ldots, \phi_p$, it holds that

$$f(x_1, \ldots, x_d) = \sum_{m=1}^p \beta_m \phi_m(x_1, \ldots, x_d), \quad (x_1, \ldots, x_d) \in \mathcal{G}_d,$$

for $\beta \in \mathbb{R}^p$. If the basis function evaluations are collected into an $n \times p$ matrix $\Phi := (\phi_m((x_1, \ldots, x_d)_i))_{i,m}$, and if the entries in the array Y are realizations of independent random variables from an exponential family as described in Appendix B, the resulting model is a GLM with design matrix Φ and regression coefficients β.

For $d \geq 2$, the d-variate basis functions can be specified via a tensor product construction in terms of d (marginal) sets of univariate functions by

$$\phi_{m_1, \ldots, m_d} := \phi_{1, m_1} \otimes \phi_{2, m_2} \otimes \cdots \otimes \phi_{d, m_d}, \quad (18)$$

where $\phi_{j,m} : \mathbb{R} \to \mathbb{R}$ for $j = 1, \ldots, d$ and $m = 1, \ldots, p_j$. The evaluation of each of the p_j univariate functions in the n_j points in $\mathcal{X}_j$ results in an $n_j \times p_j$ matrix $\Phi_j = (\phi_{j,m}(x_k))_{k,m}$. It then follows that the $n \times p$ ($p := \prod_{j=1}^d p_j$) tensor product matrix

$$\Phi = \Phi_d \otimes \cdots \otimes \Phi_1 \quad (19)$$

is identical to the design matrix for the basis evaluation in the tensor product basis, and the GLM has the structure required of a GLAM.

4.2. Benchmarking on Real Data

The multidimensional smoothing model described in the previous section was fitted using an ℓ_1-penalized B-spline basis expansion to two real datasets using the GD-PG algorithm as implemented in the R package glamlasso. See Section 6.4 for details about the R package. In this section, we report benchmark results for glamlasso and the coordinate descent-based implementation in the R package glmnet, see Friedman et al. (2010).

For both datasets, we fitted a sequence of models to data from an increasing subset of grid points, which correspond to a sequence of design matrices of increasing size. For each design matrix, we fitted 100 models for a decreasing sequence of values of the penalty parameter λ. We report the run time for fitting the sequence of 100 models using glamlasso and glmnet. We also report the run time for the combined computation of the tensor product design matrix and the fit using glmnet. The latter is more relevant for a direct comparison with glamlasso, since glamlasso requires only the marginal design matrices while glmnet requires the full tensor product design matrix.

To justify the comparison, we report the relative deviation of the objective function values attained by glamlasso from the objective function values attained by glmnet, that is,

$$\frac{F(\hat{\theta}^{\mathrm{glamlasso}}) - F(\hat{\theta}^{\mathrm{glmnet}})}{|F(\hat{\theta}^{\mathrm{glmnet}})|} \quad (20)$$

with $\hat{\theta}^{\mathrm{x}}$ denoting the estimate computed by method x. This ratio is computed for each fitted model. We note that (20) has a tendency to blow up in absolute value when F becomes small, which happens for small values of λ.

The benchmark computations were carried out on a Macbook Pro with a 2.8 GHz Intel core i7 processor and 16 GB of 1600 MHz DDR3 memory. Scripts and data are included as supplementary materials online.

4.2.1. Gaussian Neuron Data

The first dataset considered consists of spatio-temporal voltage sensitive dye recordings of a ferret brain provided by Professor Per Ebbe Roland, see Roland et al. (2006). The dataset consists of images of size 25×25 pixels recorded with a time resolution of 0.6136 ms per image. The images were recorded over 600 ms, hence the total size of this three-dimensional array dataset is $25 \times 25 \times 977$ corresponding to $n = 610{,}625$ data points.

As basis functions we used cubic B-splines with $p_j := \max\{\lceil n_j/5 \rceil, 5\}$ basis functions in each dimension (see Currie, Durban, and Eilers 2006 or Wood 2006). This corresponds to a parameter array of size $5 \times 5 \times 196$ ($p = 4900$) and a design matrix of size $610{,}625 \times 4900$ for the entire dataset. The byte size for representing this design matrix as a dense matrix was approximately 22 GB. For the benchmark, we fitted Gaussian models with the identity link function to the full dataset as well
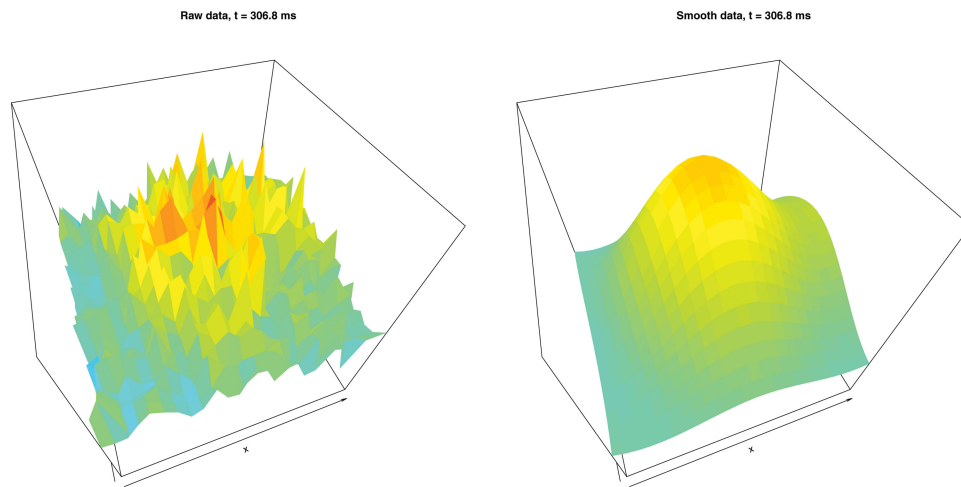
Figure 1. The raw neuron data (left) and the smoothed fit (right) after 306.8 ms. The supplementary material contains movies of the complete raw data and smoothed fit.

as to subsets of the dataset that correspond to smaller design matrices.

Figure 1 shows an example of the raw data and the smoothed fit for a particular time point. Movies of the raw data and the smoothed fit can be found as supplementary material.

Run times and relative deviations are shown in Figure 2. The model could not be fitted using `glmnet` to the full dataset due to the large size of the design matrix, and results for `glmnet` are thus only reported for models that could be fitted. The run times for `glamlasso` were generally smaller than for `glmnet`, and were, in particular, relatively insensitive to the size of the design matrix. When a sparse matrix representation of the design matrix was used, `glmnet` was able to scale to larger design matrices, but it was still clearly outperformed by `glamlasso` in terms of run time. The relative deviations in the attained objective function values were quite small.

4.2.2. Poisson Taxi Data

The second dataset considered consists of spatio-temporal information on registered taxi pickups in New York City during January 2013. The data can be downloaded from the webpage *www.andresmh.com/nyctaxitrips/*. We used a subset of this dataset consisting of triples containing longitude, latitude, and

date-time of the pickup. First, we cropped the data to pickups with longitude in $[-74.05°, -73.93°]$ and latitude in $[40.7°, 40.82°]$. Figure 3 shows the binned counts of all pickups during January 2013 with 500 bins in each spatial dimension. Pickups registered in Hudson or East River were ascribed to noise in the GPS recordings.

For this example attention was restricted to Manhattan pickups during the first week of January 2013. To this end, the data were rotated and summarized as binned counts in $100 \times 100 \times 168$ spatial-temporal bins. Each temporal bin represents 1 h. The data were then further cropped to cover Manhattan only, which removed the large black parts—as seen in Figure 3—where pickups were rare. The total size of the dataset was $33 \times 81 \times 168$ corresponding to $n = 449{,}064$ data points. The observation in each bin consisted of the integer number of pickups registered in that bin.

We used $p_j := \max\{[n_j/4], 5\}$ cubic B-spline basis functions in each dimension. The resulting parameter array was $9 \times 21 \times 42$ corresponding to $p = 7938$ and a design matrix of size $449{,}064 \times 7938$ for the entire dataset. The byte size for representing this design matrix as a dense matrix was approximately 27 GB. For the benchmark, we fitted Poisson models with the log link function to the full dataset as well as to subsets of the dataset that correspond to smaller design matrices.
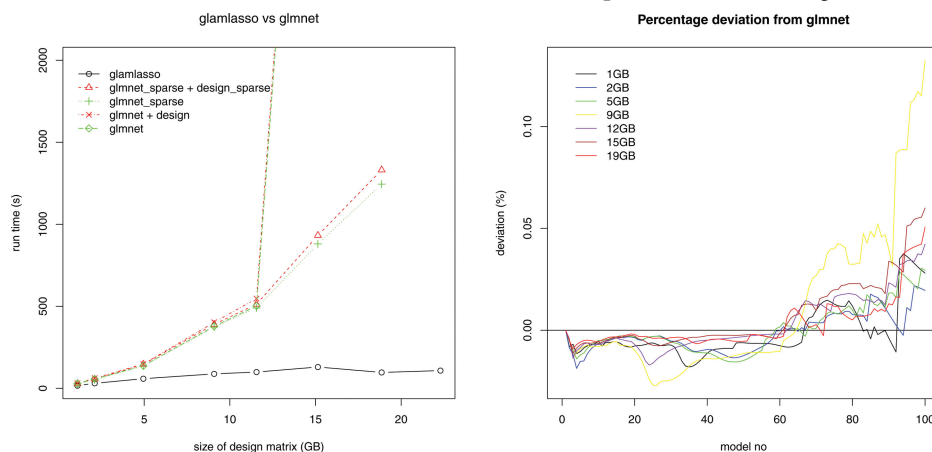
Figure 2. Benchmark results for the neuron data. Run time in seconds is shown as a function of the size of the design matrix, when not stored in sparse format, in GB (left). Relative deviation in the attained objective function values as given by (20) is shown as a function of model number (right), where a larger model number corresponds to less penalization (smaller λ).

NYC taxi pick–ups Jan. 2013

Rotated and binned data
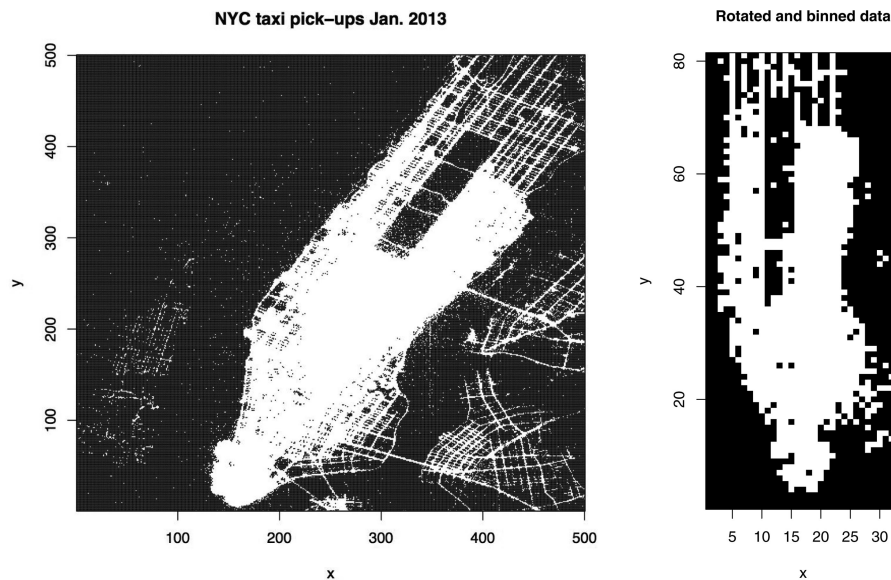
Figure 3. Binned counts of registered NYC taxi pickups for January 2013 using 500 × 500 spatial bins (left) and the same data rotated, binned to 100 × 100 spatial bins and cropped to cover Manhattan only (right).

Figure 4 shows an example of the raw data and the smoothed fit for around midnight on Saturday, January 5, 2013. Movies of the raw data and the smoothed fit can be found as supplementary material.

Run times and relative deviations are shown in Figure 5. As for the neuron data, the model could not be fitted to the full dataset using `glmnet`, and results for `glmnet` are only reported for models that could be fitted. Except for the smallest design matrix the run times for `glamlasso` were smaller than for `glmnet`, and they appear to scale better with the size of the design matrix. This was particularly so when the dense matrix representation was used with `glmnet`. The design matrix was very sparse in this example, and `glmnet` benefitted considerable in terms of run time from using a sparse storage format. The relative deviations in the attained objective function values were still acceptably small though the values attained by `glamlasso` were up to 1.5% larger than those attained by `glmnet` for the least penalized models (models fitted with small values of λ).

4.3. Using Incomplete Array Data

The implementation in `glamlasso` allows for incompletely observed arrays. This can, of course, be used for prediction of the unobserved entries by computing the smoothed fit to the incompletely observed array. In this section, we show how it can also be used for selection of the tuning parameter λ. We also refer to the supplementary materials online for scripts and data.

We used the NYC taxi data and removed the observations for 19 randomly chosen 3 × 3 blocks of spatial bins (due to overlap of some of the blocks this corresponded to 159 spatial

Raw data, t = 120 h

Smooth data, t = 120 h
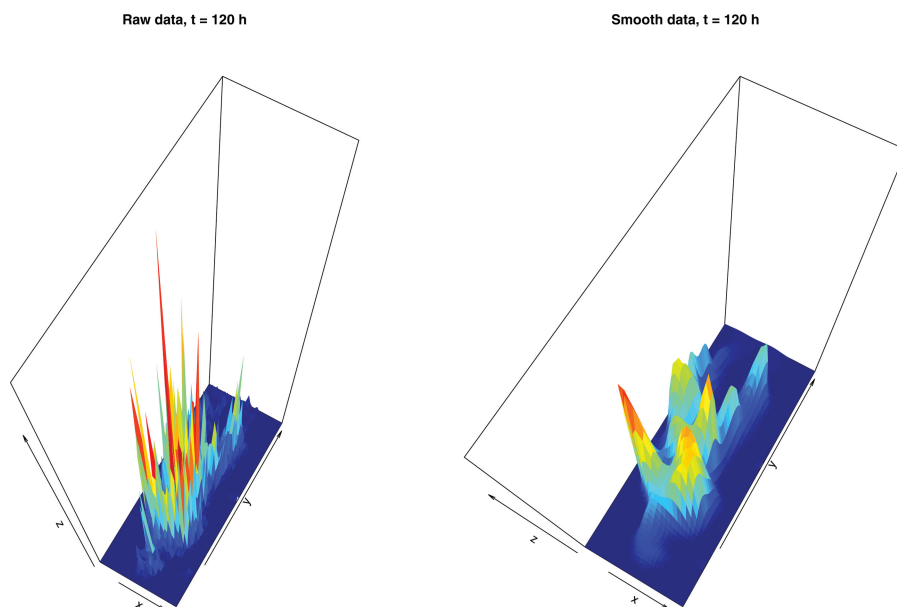
Figure 4. The raw NYC taxi data (left) and the smoothed fit (right) around midnight on Saturday, January 5, 2013. The supplementary material contains movies of the complete raw data and smoothed fit.
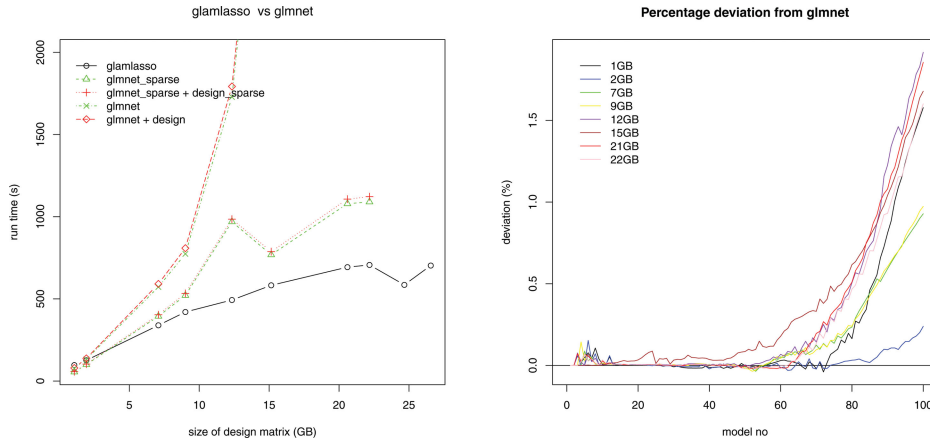
Figure 5. Benchmark results for the taxi data. Run time in seconds is shown as a function of the size of the design matrix, when not stored in sparse format, in GB (left). Relative deviations in the attained objective function values as given by (20) is shown as a function of model number (right), where a larger model number corresponds to less penalization (smaller λ).

bins). When fitting the model using `glamlasso`, the incompleteness is incorporated by setting the weights corresponding to the missing values equal to zero for all time points. We denote by D the set of grid points that correspond to the removed bins as illustrated by the red blocks in Figure 7.

From `glamlasso`, we computed a sequence of model fits corresponding to 100 values of λ, and for each value of λ we computed the fitted complete array $\hat{Y}^{(\lambda)}$ and then the mean squared error (MSE),

$$\text{MSE}(\lambda) = \sum_{x \in D} (\hat{Y}_x^{(\lambda)} - Y_x)^2,$$

as a function of λ, see Figure 6. Model 41 attained the overall minimal MSE.

Figure 7 (right panel) shows predictions for one spatial bin. The under-smoothed Model 100 gives a poor prediction while the overall optimal Model 41 gives a much better prediction.
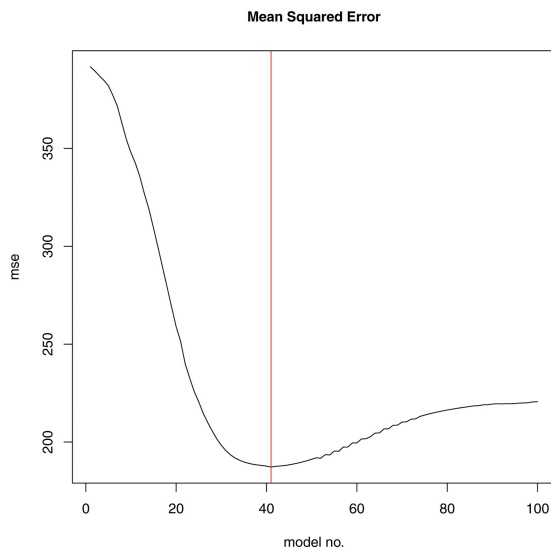
Figure 6. The mean squared error for prediction on grid points left out of the model fitting as a function of model number. The vertical red line indicates the model with minimal MSE (model 41).

5. Convergence Analysis

Our proposed GD-PG algorithm is composed of well-known components, whose convergence properties have been extensively studied. We do, however, want to clarify under which conditions the algorithm can be shown to converge and in what sense it converges. The main result in this section is a computable upper bound of the step-size, δ_k, in the inner PG loop that ensures convergence in this loop. This result hinges on the tensor product structure of the design matrix.

We first state a theorem, which follows directly from Beck and Teboulle (2010), and which for a specific choice of extrapolation sequence gives the convergence rate for the inner PG loop for minimizing the objective function

$$G := h + \lambda J, \tag{21}$$

where h is given by (15). In the following, $\|A\|_2$ denotes the spectral norm of A, which is the largest singular value of A.

Theorem 1. Let $x^* = \tilde{\theta}^{(k+1)}$ denote the minimizer defined by (10) and let the extrapolation sequence for the inner PG loop be given by $\omega_l = (l-1)/(l+2)$. Let $(x^{(l)})$ denote the sequence obtained from the inner PG loop. If $\delta^{(k)} \in (0, 1/L^{(k)}]$ where

$$L^{(k)} := \|X^\top W^{(k)} X\|_2 / n, \tag{22}$$

then

$$G(x^{(l)}) - G(x^*) \leq \frac{2L_h^{(k)} \|x^{(0)} - x^*\|_2^2}{(l+1)^2}. \tag{23}$$

Proof. The theorem is a consequence of Theorem 1.4 in Beck and Teboulle (2010) once we establish that $L^{(k)}$ is a Lipschitz constant for ∇h. To this end note that the spectral norm $\|\cdot\|_2$ is the operator norm induced by the 2-norm on $\mathbb{R}^p$, which implies that

$$\|\nabla h(\theta) - \nabla h(\theta')\|_2 \leq \frac{1}{n}\|X^\top W^{(k)} X\|_2 \|\theta - \theta'\|_2, \tag{24}$$

and $L^{(k)}$ is indeed the minimal Lipschitz constant. It should be noted that Theorem 1.4 in Beck and Teboulle (2010) is phrased in terms of an extrapolation sequence of the form $\omega_l = (t_l - 1)/t_{l+1}$ where (t_l) is a specific sequence that fulfills $t_l \geq (l+1)/2$. The extrapolation sequence considered here corresponds
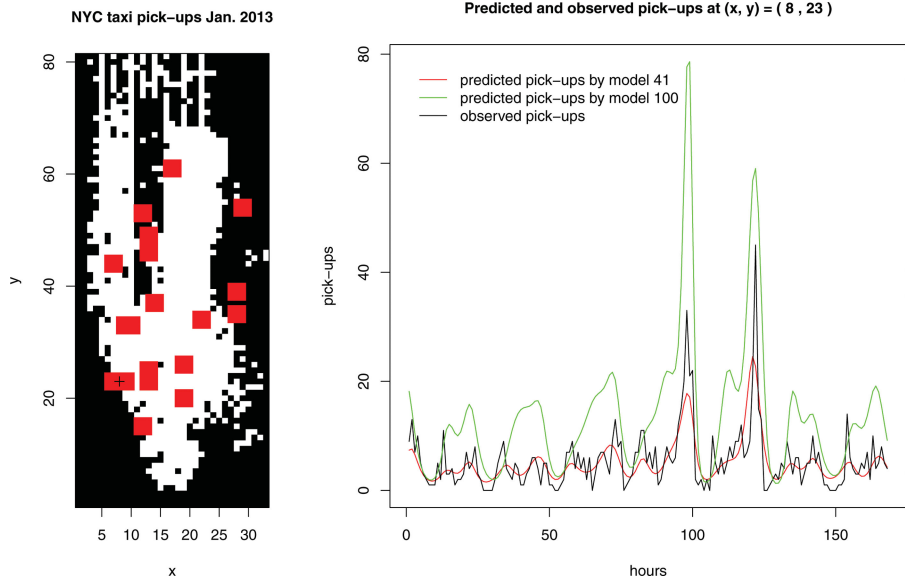
Figure 7. Binned number of NYC taxi pickups as in Figure 3 (left) with red 3 × 3 squares indicating bins that were removed from the data before model fitting. Predicted and observed number of pickups at spatial bin (8, 23) (indicated with a "+" on the left figure) are shown as a function of time in hours (right). Model 100 predictions (green) were from the least penalized model while Model 41 predictions (red) were from the model with an overall minimal MSE.

to $t_l = (l + 1)/2$, and their proof carries over to this case without changes.   □

From (23), we see that the objective function values converge at rate $O(l^{-2})$ for the given choice of extrapolation sequence. Without extrapolation, that is, with $\omega_l = 0$ for all $l \in \mathbb{N}$, the convergence rate is $O(l^{-1})$, see, for example, Theorem 1.1 in Beck and Teboulle (2010). In this case $(x^{(l)})$ always converges toward a minimizer, see Theorem 1.2 in Beck and Teboulle (2010). We are not aware of results that establish convergence of $(x^{(l)})$ for general h when extrapolation is used. However, if X has rank p and the weights are all strictly positive, the quadratic h given by (15) results in a strictly convex and level bounded objective function G, in which case (23) forces $(x^{(l)})$ to converge toward the unique minimizer.

The following result shows how the tensor product structure can be exploited to give a computable upper bound on the Lipschitz constant (22).

Proposition 1. Let $W^{(k)}$ denote the diagonal weight matrix with diagonal elements $w_i^{(k)}$, $i = 1, \ldots, n$, then

$$L^{(k)} \leq \hat{L}^{(k)} := \frac{\max(w_i^{(k)})}{n} \sum_{r=1}^{c} \prod_{j=1}^{d} \varrho(X_{r,j}^\top X_{r,j}), \quad (25)$$

where ϱ denotes the spectral radius.

Proof. Since the spectral norm is an operator norm, it is submultiplicative, which gives that

$$L^{(k)} \leq \frac{1}{n} \|X^\top\|_2 \|X\|_2 \|W^{(k)}\|_2 = \frac{1}{n} \|X\|_2^2 \|W^{(k)}\|_2. \quad (26)$$

Now $W^{(k)}$ is diagonal with nonnegative entries, so $\|W^{(k)}\|_2 = \max(w_i^{(k)})$, and $\|X\|_2^2$ is the largest eigenvalue of the symmetric matrix $X^\top X$ (the spectral radius), hence

$$L^{(k)} \leq \frac{\max(w_i^{(k)})}{n} \varrho(X^\top X). \quad (27)$$

Furthermore, as $X^\top X$ is a positive semidefinite matrix with diagonal blocks given by $X_r^\top X_r$, we get (see, e.g., Lemma 3.20 in Bapat 2010) that

$$\varrho(X^\top X) \leq \sum_{r=1}^{c} \varrho(X_r^\top X_r). \quad (28)$$

By the properties of the tensor product, we find that

$$X_r^\top X_r = X_{r,1}^\top X_{r,1} \otimes \cdots \otimes X_{r,d}^\top X_{r,d}, \quad (29)$$

whose eigenvalues are of the form $\alpha_{1,k_1} \alpha_{2,k_2} \ldots \alpha_{d,k_d}$, with α_{j,k_j} being the k_jth eigenvalue of $X_{r,j}^\top X_{r,j}$, see, for example, Theorem 4.2.12 in Horn and Johnson (1991). In particular,

$$\varrho(X_r^\top X_r) = \prod_{j=1}^{d} \varrho(X_{r,j}^\top X_{r,j}),$$

and this completes the proof.   □

Note that for $c = 1$, the upper bound is $\hat{L}^{(k)} = \max(w_i^{(k)}) \prod_{j=1}^{d} \varrho(X_{1,j}^\top X_{1,j})/n$, which is valid for any weight matrix. If the weight matrix is itself a tensor product, it is possible to compute the Lipschitz constant exactly. Indeed, if $W^{(k)} = W_d^{(k)} \otimes \cdots \otimes W_1^{(k)}$ then

$$X^\top W^{(k)} X = X_{1,d}^\top W_d^{(k)} X_{1,d} \otimes \cdots \otimes X_{1,1}^\top W_1^{(k)} X_{1,1},$$

and by similar arguments as in the proof above,

$$L^{(k)} = \frac{1}{n} \prod_{j=1}^{d} \varrho(X_{1,j}^\top W_j^{(k)} X_{1,j}). \quad (30)$$

The outer loop is similar to the outer loop used in, for example, the R packages `glmnet`, Friedman et al. (2010), and `sglOptim`, Vincent and Hansen (2014). For completeness, we demonstrate that the outer loop with the stepsize determined by the Armijo rule is a special case of the algorithm treated in Tseng

and Yun (2009), which implies a global convergence result of the outer loop.

Following Tseng and Yun (2009), the Armijo rule gives the stepsize $\alpha_k := b^j \alpha_0$, where $\alpha_0 > 0$ and $b \in (0, 1)$ are given constants and j is determined as follows: With $d^{(k)} = \tilde{\theta}^{(k+1)} - \theta^{(k)}$ and

$$\Delta_k := -(u^{(k)})^\top X d^{(k)} + \lambda (J(\tilde{\theta}^{(k+1)}) - J(\theta^{(k)})),$$

then $j \in \mathbb{N}_0$ is the smallest nonnegative integer for which

$$F(\theta^{(k)} + b^j \alpha_0 d^{(k)}) \leq F(\theta^{(k)}) + b^j \alpha_0 \upsilon \Delta_k, \tag{31}$$

where $\upsilon \in (0, 1)$ is a fixed constant.

Theorem 2. Let the stepsize, α_k, be given by the Armijo rule above. If the design matrix X has rank p and if there exist constants $\bar{c} \geq \underline{c} > 0$ such that for all $k \in \mathbb{N}$ the diagonal weights in $W^{(k)}$, denoted $w_i^{(k)}$, satisfy

$$\underline{c} \leq w_i^{(k)} \leq \bar{c} \tag{32}$$

for $i = 1, \ldots, n$, then $(F(\theta^{(k)}))$ is nonincreasing and any cluster point of $(\theta^{(k)})$ is a stationary point of the objective function F.

Proof. The theorem is a consequence of Theorem 1 (a) and (e) in Tseng and Yun (2009) once we have established that the search direction, $d^{(k)} = \tilde{\theta}^{(k+1)} - \theta^{(k)}$, coincides with the search direction defined by (6) in Tseng and Yun (2009). Letting $d := \theta - \theta^{(k)}$ denote a (potential) search direction we see that

$$\frac{1}{2n} \| \sqrt{W^{(k)}} (X\theta - z^{(k)}) \|_2^2$$
$$= \frac{1}{2n} (-(W^{(k)})^{-1} u^{(k)}$$
$$\quad + X(\theta - \theta^{(k)}))^\top W^{(k)} (-(W^{(k)})^{-1} u^{(k)} + X(\theta - \theta^{(k)}))$$
$$= \frac{1}{2n} ((u^{(k)})^\top (W^{(k)})^{-1} u^{(k)} - (u^{(k)})^\top X d - d^\top X^\top u^{(k)}$$
$$\quad + d^\top X^\top W^{(k)} X d)$$
$$\propto - \underbrace{(u^{(k)})^\top X}_{\nabla_\theta l(\eta^{(k)})^\top} d + \frac{1}{2} d^\top \underbrace{X^\top W^{(k)} X}_{H^{(k)}} d + C_k,$$

where C_k is a constant not depending upon θ. This shows that

$$d^{(k)} = \underset{d \in \mathbb{R}^p}{\arg\min} - \nabla_\theta l(\eta^{(k)})^\top d + \frac{1}{2} d^\top H^{(k)} d + \lambda J(\theta^{(k)} + d),$$
$$\tag{33}$$

and this is indeed the search direction defined by (6) in Tseng and Yun (2009) (with the coordinate block consisting of all coordinates). Observe that $H^{(k)} = X^T W^{(k)} X$ fulfills Assumption 1 in Tseng and Yun (2009) by the assumptions that X has rank p and that the weights are uniformly bounded away from 0 and ∞. Therefore, all conditions for Theorem 1 in Tseng and Yun (2009) are fulfilled, which completes the proof. □

The convergence conclusion can be sharpened by making further assumptions on the objective function and the weights.

Corollary 1. Suppose that the weights are given by

$$w_i^{(k)} = \vartheta'(\eta_i^{(k)})(g^{-1})'(\eta_i^{(k)}), \quad i = 1, \ldots, n. \tag{34}$$

If X has rank p, if F is level bounded, if the PMLE, θ^*, is unique and if $(g^{-1})'$ is nonzero everywhere it holds that $\theta^{(k)} \to \theta^*$ for $k \to \infty$.

Proof. The sublevel set $\Theta_0 := \{ \theta \mid F(\theta) \leq F(\theta^{(0)}) \}$ is bounded by assumption, and it is closed because J is closed and $-l$ is continuous. Hence, Θ_0 is compact. Since the weights as a function of θ,

$$\theta \mapsto \vartheta'(\eta_i(\theta))(g^{-1})'(\eta_i(\theta)) \tag{35}$$

for $i = 1, \ldots, n$, are continuous and strictly positive functions—because $(g^{-1})'$ is assumed nonzero everywhere, see Appendix B—they attain a strictly positive minimum and a finite maximum over the compact set Θ_0. This implies that (32) holds. Since $\theta^{(k)} \in \Theta_0$ and θ^* is a unique stationary point in Θ_0, it follows from Theorem 2, using again that Θ_0 is compact, that $\theta^{(k)} \to \theta^*$ for $k \to \infty$. □

The weights given by (34) are the common weights used for GLMs, but exactly the same argument as above applies to other choices as long as they are strictly positive and continuous functions of the parameter θ. A notable special case is $w_i^{(k)} = 1$. Another possibility, which is useful in the framework of GLAMs, is discussed in Section 6.

Observe that if $-l$ is strongly convex, then F is level bounded, X has rank p and θ^* is unique. If X does not have rank p, in particular, if $p > n$, we are not presenting any results on the global convergence of the outer loop. Clearly, additional assumptions on the penalty function J must then be made to guarantee convergence.

6. Implementation

In this section, we show how the computations required in the GD-PG algorithm can be implemented to exploit the array structure. The penalty function J is not assumed to have any special structure in general, and its evaluation is not discussed, but we do briefly discuss the computation of the proximal operator for some special choices of J. We also describe the R package, glamlasso, which implements the algorithm for two- and three-dimensional array models with the ℓ_1-penalty and the smoothly clipped absolute deviation (SCAD) penalty, and we present results of further benchmark studies using simulated data.

6.1. Array Operations

The linear algebra operations needed in the GD-PG algorithm can all be expressed in terms of two maps, H and G, which are defined below. The maps work directly on the tensor factors in terms of ρ defined in Appendix A. Introduce

$$H(\langle X_{r,j} \rangle, \langle \Theta_r \rangle) := \sum_{r=1}^c \rho(X_{r,d}, \ldots, \rho(X_{r,1}, \Theta_r) \ldots), \tag{36}$$

which gives an $n_1 \times \cdots \times n_d$ array such that $\text{vec}(H(\langle X_{r,j} \rangle, \langle \Theta_r \rangle))$ is the linear predictor. Introduce also

$$G(\langle X_{r,j} \rangle, U) := \langle \rho(X_{1,d}^\top, \ldots, \rho(X_{1,1}^\top, U) \ldots), \ldots,$$
$$\rho(X_{c,d}^\top, \ldots, \rho(X_{c,1}^\top, U) \ldots) \rangle \tag{37}$$

for U an $n_1 \times \cdots \times n_d$ array, which gives a tuple of c arrays. The map G is used to carry out the gradient computation in (4).

In the following we describe how the linear algebra operations required in Steps 2, 4, and 5 in Algorithm 1 can be carried out using the two maps above. In doing so we use "$\equiv$" to denote equality of vectors and arrays (or tuples of arrays) up to a rearrangement of the entries. In the implementation such a rearrangement is never required, but it gives a connection between the array and vector representations of the components in the algorithm.

Step 2: The linear predictor is first computed,

$$X^\top \theta^{(k)} \equiv \mathrm{H}(\langle X_{r,j} \rangle, \langle \Theta_r^{(k)} \rangle). \tag{38}$$

The array $V^{(k)}$ is computed by an entrywise computation, for example, by (34). The arrays $U^{(k)}$ and $Z^{(k)}$ are computed by entrywise computations using (B.2) and (9), respectively. If the weights given by (34) are used, $Z^{(k)}$ can be computed directly by (B.4) and $U^{(k)}$ does not need to be computed.

Step 4: In the inner PG loop, the gradient, ∇h, must be recomputed in each iteration. To this end,

$$X^\top W^{(k)} z^{(k)} \equiv \mathrm{G}(\langle X_{r,j} \rangle, V^{(k)} \odot Z^{(k)}) \tag{39}$$

is precomputed. Here, $\odot$ denotes the entrywise (Hadamard) product. Then $\nabla h(\theta)$ is computed in terms of

$$X^\top W^{(k)} X \theta \equiv \mathrm{G}(\langle X_{r,j} \rangle, V^{(k)} \odot \mathrm{H}(\langle X_{r,j} \rangle, \langle \Theta_r \rangle)). \tag{40}$$

Step 5: For the stepsize computation using the Armijo rule the linear predictor,

$$X^\top \tilde{\theta}^{(k+1)} \equiv \mathrm{H}(\langle X_{r,j} \rangle, \langle \tilde{\Theta}_r^{(k+1)} \rangle), \tag{41}$$

is first computed. The computation of Δ_k is achieved via computing inner products of $U^{(k)}$ and the linear predictors (38) and (41). The line search then involves iterative recomputations of the linear predictor via the map H.

If δ_k is not chosen sufficiently small to guarantee convergence of the inner PG loop, a line search must also be carried out in Step 4. To this end, repeated evaluations of h are needed, with $h(\theta)$ being computed as the weighted 2-norm of $\mathrm{H}(\langle X_{r,j} \rangle, \langle \Theta_r \rangle) - Z^{(k)}$ with weights $V^{(k)}$.

6.2. Tensor Product Weights

The bottleneck in the GD-PG algorithm is (40), which is an expensive operation that has to be carried out repeatedly. If the diagonal weight matrix is a tensor product, the computations can be organized differently. This can reduce the run time, especially when $p_{r,j} < n_j$.

Suppose that $W^{(k)} = W_d^{(k)} \otimes \cdots \otimes W_1^{(k)}$, then

$$X_r^\top W^{(k)} X_m = X_{r,d}^\top W_d^{(k)} X_{m,d} \otimes \cdots \otimes X_{r,1}^\top W_1^{(k)} X_{m,1},$$
$$r, m = 1, \dots, c.$$

Hence $X^\top W^{(k)} X$ has tensor product blocks and (40) can be replaced by

$$X^\top W^{(k)} X \theta \equiv \langle \mathrm{H}(\langle X_{1,j}^\top W_j^{(k)} X_{r,j} \rangle, \langle \Theta_r \rangle),$$
$$\dots, \mathrm{H}(\langle X_{c,j}^\top W_j^{(k)} X_{r,j} \rangle, \langle \Theta_r \rangle) \rangle. \tag{42}$$

The matrix products $X_{r,k}^\top W_j^{(k)} X_{m,j}$ for $r, m = 1, \dots, c$ and $j = 1, \dots, d$ can be precomputed in Step 4.

If the weight matrix is not a tensor product, it might be approximated by one so that (42) can be exploited. With $V^{(k)}$ denoting the weights in array form, then $V^{(k)}$ can be approximated by $\hat{V}^{(k)}$, where

$$\hat{V}_{i_1,\dots,i_d}^{(k)} = \hat{v}_{1,i_1}^{(k)} \cdots \hat{v}_{d,i_d}^{(k)}, \tag{43}$$

with

$$\hat{v}_{j,i_j}^{(k)} = \left(\prod_{i_1,\dots,i_{j-1},i_{j+1},\dots,i_d} \frac{V_{i_1,\dots,i_d}^{(k)}}{\overline{V}^{(k)}} \right)^{\frac{1}{m_j}}$$

$$= \exp\left(\frac{1}{m_j} \sum_{i_1,\dots,i_{j-1},i_{j+1},\dots,i_d} \log V_{i_1,\dots,i_d}^{(k)} - \log \overline{V}^{(k)} \right).$$

Here $m_j = n/n_j = \prod_{j' \neq j} n_{j'}$ and

$$\overline{V}^{(k)} = \left(\prod_{i_1,\dots,i_d} V_{i_1,\dots,i_d} \right)^{\frac{1}{n}}.$$

The array $\hat{V}^{(k)}$ is equivalent to a diagonal weight matrix, which is a tensor product of diagonal matrices with diagonals $(\hat{v}_{j,i}^{(k)})$. Observe that if the weights in $V^{(k)}$ satisfy (32) then so do the approximating weights in $\hat{V}^{(k)}$.

6.3. Proximal Operations

Efficient computation of the proximal operator is necessary for the inner PG loop to be fast. Ideally $\mathrm{prox}_\gamma(z)$ should be given in a closed form that is fast to evaluate. This is the case for several commonly used penalty functions such as the 1-norm, the squared 2-norm, their linear combination, and several other separable penalty functions.

For the 1-norm, $\mathrm{prox}_\gamma(z)$ is given by soft thresholding, see Beck and Teboulle (2010) or Parikh and Boyd (2014), that is,

$$\mathrm{prox}_\gamma(z)_i = (|z_i| - \gamma)_+ \mathrm{sign}(z_i). \tag{44}$$

For the squared 2-norm (ridge penalty), the proximal operator amounts to multiplicative shrinkage,

$$\mathrm{prox}_\gamma(z) = \frac{1}{1 + 2\gamma} z, \tag{45}$$

see, for example, Moreau (1962). For the elastic net penalty,

$$J(\theta) = ||\theta||_1 + \alpha ||\theta||_2^2, \tag{46}$$

the proximal operator amounts to a composition of the proximal operators for the 1-norm and the squared 2-norm, that is,

$$\mathrm{prox}_\gamma(z)_i = \frac{1}{1 + 2\alpha\gamma} (|z_i| - \gamma)_+ \mathrm{sign}(z_i), \tag{47}$$

see Parikh and Boyd (2014). For more examples see Parikh and Boyd (2014) and see also Zhang, Jiang, and Luo (2013) for the proximal group shrinkage operator.

6.4. The `glamlasso` R Package

The `glamlasso` R package provides an implementation of the GD-PG algorithm for ℓ_1-penalized as well as SCAD-penalized estimation in two- and three-dimensional GLAMs. We note that as the SCAD penalty is nonconvex the resulting optimization problem becomes nonconvex and hence falls outside the original scope of our proposed method. However, by a local linear approximation to the SCAD penalty, one obtains a weighted ℓ_1-penalized problem. This is a convex problem, which may be solved within the framework proposed above. Especially, by iteratively solving a sequence of appropriately weighted ℓ_1-penalized problems, it is, in fact, possible to solve nonconvex problems, see Zou and Li (2008). In the `glamlasso` package, this is implemented using the multistep adaptive lasso (MSA-lasso) algorithm from Bühlmann and van de Geer (2011).

The package is written in C++ and uses the `Rcpp` package for the interface to R, see Eddelbuettel and François (2011). At the time of writing, this implementation supports the Gaussian model with identity link, the Binomial model with logit link, the Poisson model with log link, and the Gamma model with log link, but see Lund (2016) for the current status.

The function `glamlasso` in the package solves the problem (5) with J either given by the ℓ_1-penalty or the SCAD penalty for a (user-specified) number of penalty parameters $\lambda_{\max} > \cdots > \lambda_{\min}$. Here $\lambda_{\max}$ is the infimum over the set of penalty parameters yielding a zero solution to (5) and $\lambda_{\min}$ is a (user-specified) fraction of $\lambda_{\max}$. For each model (λ-value), the algorithm is warm-started by initiating the algorithm at the solution for the previous model.

The interface of the function `glamlasso` resembles that of the `glmnet` function with some GD-PG-specific options.

The argument `penalty` controls the type of penalty to use. Currently, the ℓ_1-penalty (`"lasso"`) and the SCAD penalty (`"scad"`) are implemented.

The argument `steps` controls the number of steps to use in the MSA algorithm when the SCAD penalty is used.

The argument $\nu \in [0, 1]$ (nu) controls the stepsize in the inner PG loop relative to the upper bound, $\hat{L}^{(k)}$, on the Lipschitz constant. Especially, for $\nu \in (0, 1)$ the stepsize is initially $\delta^{(k)} := 1/(\nu \hat{L}^{(k)})$ and the backtracking procedure from Beck and Teboulle (2009) is employed only if divergence is detected. For $\nu = 1$, the stepsize is $\delta^{(k)} := 1/\hat{L}_h$ and no backtracking is done. For $\nu = 0$, the stepsize is initially $\delta^{(k)} := 1$ and backtracking is done in each iteration.

The argument `iwls = c("exact", "one", "kron1", "kron2" )` specifies whether a tensor product approximation to the weights or the exact weights are used. The exact weights are the weights given by (34). Note that while a tensor product approximation may reduce the run time for the individual steps in the inner PG loop, it may also affect the convergence of the entire loop negatively.

Finally, the argument `Weights` allows for a specification of observation weights. This can be used—as mentioned in Currie, Durban, and Eilers (2006)—as a way to model scattered (nongrid) data using a GLAM by binning the data and then weighing each bin according to the number of observations in the bin. By setting some observation weights to 0, it is also possible to model incompletely observed arrays as illustrated in Section 4.3.

6.5. Benchmarking on Simulated Data

To further investigate the performance of the GD-PG algorithm and its implementation in `glamlasso`, we carried out a benchmark study based on simulated data from a three-dimensional GLAM. We report the setup and the results of the benchmark study in this section. See the supplementary materials online for scripts used in this section.

For each $j \in \{1, 2, 3\}$, we generated an $n_j \times p_j$ matrix X_j by letting its rows be n_j independent samples from an $\mathcal{N}_{p_j}(0, \Sigma)$ distribution. The diagonal entries of the covariance matrix Σ were all equal to $\sigma > 0$ and the off diagonal elements were all equal to κ for different choices of κ. Since the design matrix $X = X_3 \otimes X_2 \otimes X_1$ is a tensor product, there is a nonzero correlation between the columns of X even when $\kappa = 0$. Furthermore, each column of X contains n samples from a distribution with density given by a Meijer G-function, see Springer and Thompson (1970).

We considered designs with $n_1 = 60r$, $n_2 = 20r$, $n_3 = 10r$ and $p_1 = \max\{3, n_1 q\}$, $p_2 = \max\{3, n_2 q\}$, $p_3 = \max\{3, n_3 q\}$ for a sequence of r-values and $q \in \{0.5, 3\}$. The number q controls if $p < n$ or $p > n$ and the size of the design matrix increases with r.

The regression coefficients were generated as

$$\theta_m = (-1)^m \exp\left(\frac{-(m-1)}{10}\right) B_m, \quad m = 1, \ldots, p,$$

where $B_1, \ldots, B_p$ are iid Bernoulli variables with $P(B_m = 1) = s$ for $s \in [0, 1]$. Note that s controls the sparsity of the coefficient vector and $s = 1$ results in a dense parameter vector.

We generated observations from two different models for different choices of parameters.

Gaussian models: We generated Gaussian observations with unit variance and the identity link with a dense parameter vector ($s = 1$). The design was generated with $\sigma = 1$ and $\kappa \in \{0, 0.25\}$ for $p < n$ and $\kappa = 0$ for $p > n$.

Poisson models: We generated Poisson observations with the log link function with a sparse parameter vector ($s = 0.01$). The design was generated with $\sigma = 0.71$ and $\kappa \in \{0, 0.25\}$ for $p < n$ and $\kappa = 0$ for $p > n$. It is worth noting that this quite artificial Poisson simulation setup easily generates extremely large observations, which in turn can cause convergence problems for the algorithms, or even NA values.

For each of the two models above and for the different combinations of design and simulation parameters, we computed the PMLE using `glamlasso` as well as `glmnet` for the same sequence of λ-values. The default length of this sequence is 100,
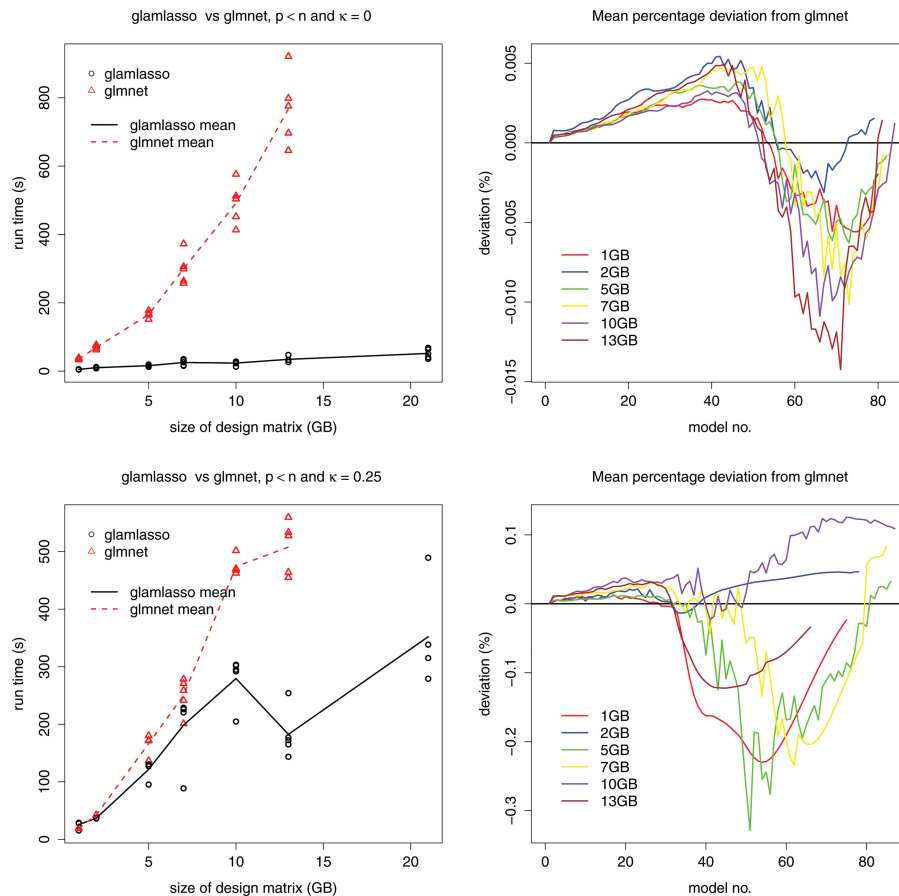
Figure 8. Benchmark results for the Gaussian models and $p < n$. Run time in seconds is shown as a function of the size of the design matrix in GB (left). Relative mean deviation in the attained objective function values as given by (20) is shown as a function of model number (right). The top row gives the results for $\kappa = 0$ and the bottom for $\kappa = 0.25$.

however, both `glmnet` and `glamlasso` will exit if convergence is not obtained for some λ value and return only the PMLEs for the preceding models along with the corresponding λ sequence.

This benchmark study on simulated data was carried out on the same computer as used for the benchmark study on real data as presented in Section 4.2 However, here we ran the simulation and optimization procedures five times for each size and parameter combination and report the run times along with their means as well as the mean relative deviations of the objective functions. See Section 4.2 for other details on how `glamlasso` and `glmnet` were compared. Figures 8–10 present the results.

Figure 8 shows the results for the Gaussian models for $p < n$. Here `glamlasso` generally outperformed `glmnet` in terms of run time—especially for $\kappa = 0$. It scaled well with the size of the design matrix and it could fit the model for large design matrices that `glmnet` could not handle.

It should be noted that for the Gaussian models with the identity link there is no outer loop, hence the comparison is in this case effectively between the (GLAM enhanced) proximal gradient algorithm and the coordinate descent algorithm as implemented in `glmnet`.

Figure 9 shows the results for the Poisson models for $p < n$. As for the Gaussian case, `glamlasso` was generally faster than `glmnet`. The run times for `glamlasso` also scaled very well with the size of the design matrix for both values of κ.

Figure 10 shows the results for both models for $p > n$ and $\kappa = 0$. Here the run times were comparable for small design matrices, with `glmnet` being a little faster for the Gaussian model, but `glamlasso` still scaled better with the size of the design matrix. For $\kappa > 0$ (results not shown), `glamlasso` retained its benefit in terms of memory usage, but `glmnet` became comparable or even faster for the Gaussian model than `glamlasso`.

In the comparisons above we have not included the time it took to construct the actual design matrix for the `glmnet` procedure. However, the construction and handling of matrices, whose size is a substantial fraction of the computers' memory, was quite time consuming (between 15 min and up to 1 h) underlining the advantage of our design matrix free method.

7. Discussion

The algorithm implemented in the R package `glmnet` and described in Friedman et al. (2010) computes the penalized and weighted least-square estimate given by (10) by a coordinate descent algorithm. For penalty functions like the 1-norm that induce sparsity of the minimizer, this is recognized as a very efficient algorithm. Our initial strategy was to adapt the coordinate descent algorithm to GLAMs so that it could take advantage of the tensor product structure of the design matrix. It turned out to be difficult to do that. It is straightforward to implement a memory efficient version of the coordinate descent algorithm
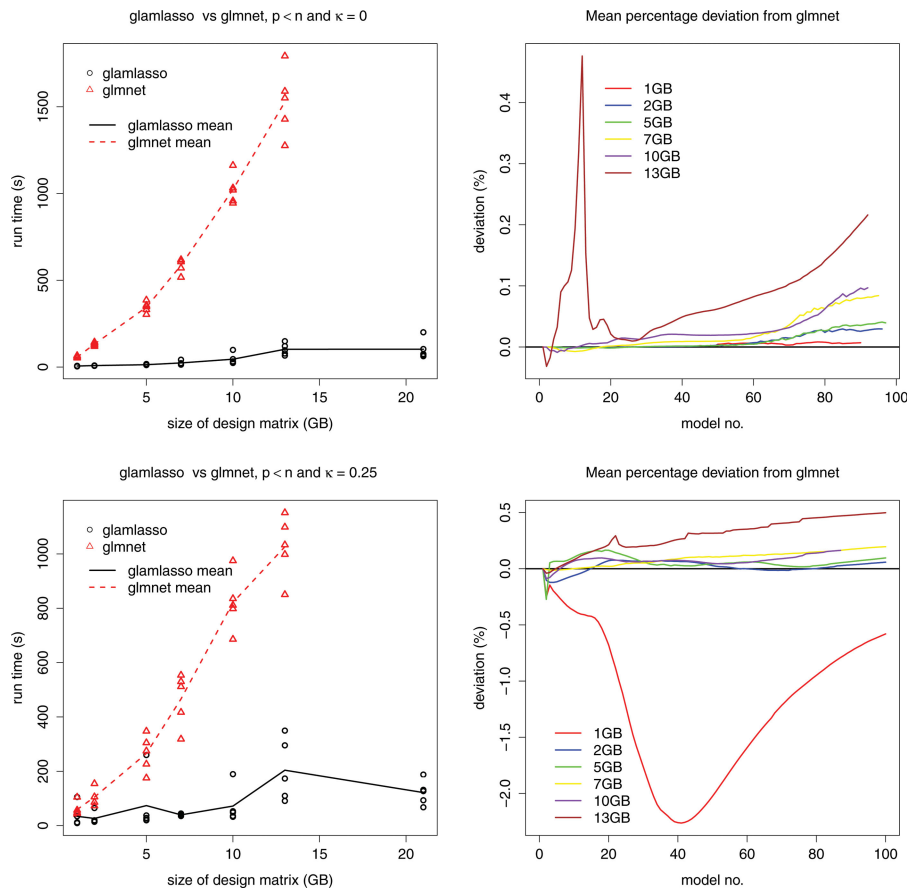
Figure 9. Benchmark results for the Poisson models and $p < n$. Run time in seconds is shown as a function of the size of the design matrix in GB (left). Relative mean deviation in the attained objective function values as given by (20) is shown as a function of model number (right). The top row gives the results for $\kappa = 0$ and the bottom for $\kappa = 0.25$.

that does not require the storage of the full tensor product design matrix, but it is not obvious how to exploit the array structure to reduce the computational complexity. Consequently, our implementation of such an algorithm was outperformed by `glmnet` in terms of run time, and for this reason alternatives to the coordinate descent algorithm were explored.

Proximal gradient algorithms for solving nonsmooth optimization problems have recently received renewed attention. One reason is that they have shown to be useful for large-scale data analysis problems, see, for example, Parikh and Boyd (2014). In the image analysis literature, the proximal gradient algorithm for a squared error loss with an ℓ_1-penalty is known as ISTA (iterative selection-thresholding algorithm), see Beck and Teboulle (2009) and Beck and Teboulle (2010). The accelerated version with a specific extrapolation sequence was dubbed FISTA (fast ISTA) by Beck and Teboulle (2009). For small-scale problems and unstructured design matrices, it is our experience that the coordinate descent algorithm outperforms accelerated proximal algorithms like FISTA. This observation is also in line with the more systematic comparisons presented in sec. 5.5 in Hastie, Tibshirani, and Wainwright (2015). For large-scale problems and/or structured design matrices—such as the tensor product design matrices considered in this article—the proximal gradient algorithms may take advantage of the structure. The Gaussian smoothing example demonstrated that this is indeed the case.

When the squared error loss is replaced by the negative log-likelihood, our proposal is similar to the approach taken in

`glmnet`, where penalized weighted least-square problems are solved iteratively by an inner loop. The main difference is that we suggest using a proximal gradient algorithm instead of a coordinate descent algorithm for the inner loop. Including weights is only a trivial modification of FISTA from Beck and Teboulle (2009), but the weight matrix commonly used for fitting GLMs is not a tensor product. Despite this, it is still possible to exploit the tensor product structure to speed up the inner loop, but by making a tensor approximation to the weights we obtained in some cases further improvements. For this reason, we developed the GD-PG algorithm with an arbitrary choice of weights. The Poisson smoothing example demonstrated that when compared to coordinate descent, the inner PG loop was capable of taking advantage of the tensor product structure.

The convergence analysis combines general results from the optimization literature to obtain convergence results for the inner proximal algorithm and the outer gradient-based descent algorithm. These results are strongest when the design matrix has rank p (thus requiring $p \leq n$). Convergence for $p > n$ would require additional assumptions on J, which we have not explored in any detail. Our experience for $J = \| \cdot \|_1$ is that the algorithm converges in practice also when $p > n$. Our most important contribution to the convergence analysis is the computation of the upper bound $\hat{L}^{(k)}$ of the Lipschitz constant $L^{(k)}$. This upper bound relies on the tensor product structure. For large-scale problems, the computation of $L^{(k)}$ will in general be infeasible due to the size of $X^\top W^{(k)} X$. However, for the tensor product design matrices considered, the upper bound is computable, and
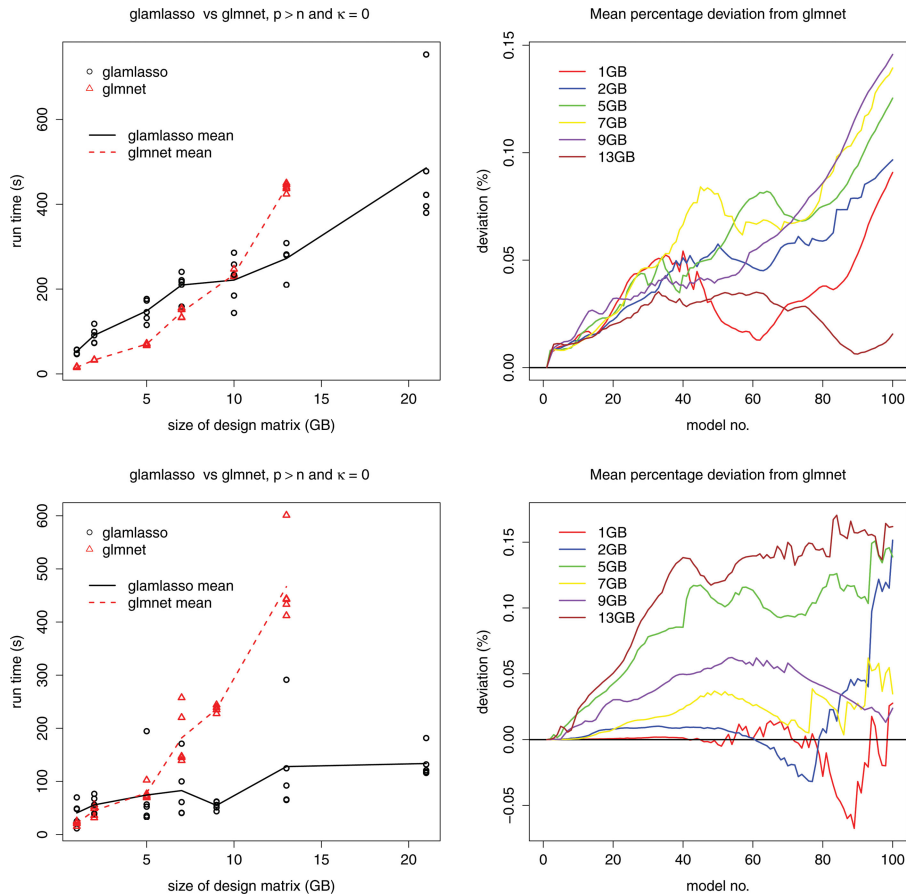
Figure 10. Benchmark results for $p > n$. Run time in seconds is shown as a function of the size of the design matrix in GB (left). Relative mean deviation in the attained objective function values as given by (20) is shown as a function of model number (right). The top row gives the results for the Gaussian model and the bottom for Poisson model.

a permissible stepsize $\delta^{(k)}$ that ensures convergence of the inner PG loop can be chosen.

It should be noted that the GD-PG algorithm requires minimal assumptions on J, but that the proximal operator associated with J should be fast to compute for the algorithm to be efficient. Though it has not been explored in this article, the generality allows for the incorporation of convex parameter constraints. For box constraints J will be separable and the proximal operator will be fast to compute.

The simulation study confirmed what the smoothing applications had showed, namely, that the GD-PG algorithm with $J = \| \cdot \|_1$ and its implementation in the R package `glamlasso` scales well with the problem size. It can, in particular, efficiently handle problems where the design matrix becomes prohibitively large to be computed and stored explicitly. Moreover, in the simulation study the run times were in most cases smaller than or comparable to that of `glmnet` even for small problem sizes. However, the simulation study also revealed that when $p > n$, the run time benefits of `glamlasso` over `glmnet` were small or diminished completely—in particular for small problem sizes. One explanation could be that `glmnet` implements a screening rule, which is particularly beneficial when $p > n$. It appears to be difficult to combine such screening rules with the tensor product structure of the design matrix. When $p < n$, as in the smoothing applications, `glamlasso` was, however, faster than `glmnet` and scaled much better with the size of the

problem. This was true even when a sparse representation of the design matrix was used, though `glmnet` was faster and scaled better with the size of the design matrix in this case for both examples. It should be noted that `glamlasso` achieves its performance without relying on sparsity of the design matrix, and it thus works equally well for smoothing with nonlocal as well as local basis functions.

In conclusion, we have developed and implemented an algorithm for computing the penalized maximum likelihood estimate for a GLAM. When compared to Currie, Durban, and Eilers (2006), our focus has been on nonsmooth penalty functions that yield sparse estimates. It was shown how the proposed GD-PG algorithm can take advantage of the GLAM data structure, and it was demonstrated that our implementation is both time and memory efficient. The smoothing examples illustrated how GLAMs can easily be fitted to 3D data on a standard laptop computer using the R package `glamlasso`.

Appendix A: The Maps vec and ρ

The map vec maps an $n_1 \times \cdots \times n_d$ array to a $\prod_{i=1}^{d} n_d$-dimensional vector. This is sometimes known as "flattening" the array. For $j = 1, \dots, d$ and $i_j = 1, \dots, n_j$ introduce the integer

$$[i_1, \dots, i_d] := i_1 + n_1((i_2 - 1) + n_2((i_3 - 1) + \cdots n_{d-1}(i_d - 1)\dots)).$$

$$(A.1)$$

Then vec is defined as

$$\text{vec}(A)_{[i_1,\ldots,i_d]} := A_{i_1,\ldots,i_d} \tag{A.2}$$

for an array A. This definition of vec corresponds to flattening a matrix in column-major order.

Following the definitions in Currie, Durban, and Eilers (2006) (see also De Boor 1979; Buis and Dyksen 1996), ρ maps an $r \times n_1$ matrix and an $n_1 \times \cdots \times n_d$ array to an $n_2 \times \cdots \times n_d \times r$ array. With X the matrix and A the array then

$$\rho(X, A)_{i_1,\ldots,i_d} := \sum_j X_{i_d, j} A_{j, i_1,\ldots,i_{d-1}}. \tag{A.3}$$

From this definition, it follows directly that

$$
\begin{aligned}
(X_d & \otimes \cdots \otimes X_1)\,\text{vec}(A)_{[i_1,\ldots,i_d]} \\
&= \sum_{j_1,\ldots,j_d} X_{d,i_d,j_d} \ldots X_{1,i_1,j_1} A_{j_1,\ldots,j_d} \\
&= \sum_{j_d} X_{d,i_d,j_d} \cdots \sum_{j_2} X_{2,i_2,j_2} \sum_{j_1} X_{1,i_1,j_1} A_{j_1,\ldots,j_d} \\
&= \rho(X_d, \ldots, \rho(X_2, \rho(X_1, A)) \ldots)_{i_1,\ldots,i_d}
\end{aligned}
$$

where $[i_1,\ldots,i_d]$ denotes the index defined by (A.1).

Appendix B: Exponential Families

The exponential families considered are distributions on $\mathbb{R}$ whose density is

$$f_{\vartheta,\psi}(y) = \exp\left(\frac{a(\vartheta y - b(\vartheta))}{\psi}\right)$$

w.r.t. some reference measure. Here ϑ is the canonical (real valued) parameter, $\psi > 0$ is the dispersion parameter, $a > 0$ is a known and fixed weight and b is the log-normalization constant as a function of ϑ that ensures that the density integrates to 1. In general, ϑ may have to be restricted to an interval depending on the reference measure used. Note that the reference measure will depend upon ψ but not on ϑ.

With η denoting the linear predictor in a generalized linear model, we regard $\vartheta(\eta)$ as a parameter function that maps the linear predictor to the canonical parameter, such that the mean equals $g^{-1}(\eta)$ when g is the link function. From this it can easily be derived that $b'(\vartheta(\eta)) = g^{-1}(\eta)$. For a canonical link function, $\vartheta(\eta) = \eta$ and $b' = g^{-1}$. In terms of η the log-density can be written as

$$\log f_{\vartheta(\eta),\psi}(y) \propto a(\vartheta(\eta)y - b(\vartheta(\eta))).$$

From this it follows that

$$\partial_\eta \log f_{\vartheta(\eta),\psi}(y) = a\vartheta'(\eta)(y - g^{-1}(\eta)), \tag{B.1}$$

and the score statistic, $u = \nabla_\eta l(\eta)$, entering in (4) is thus given by

$$u_i = a_i \vartheta'(\eta_i)(y_i - g^{-1}(\eta_i)), \quad i = 1, \ldots n. \tag{B.2}$$

The weights commonly used when fitting a GLM are

$$w_i = \vartheta'(\eta_i)(g^{-1})'(\eta_i), \tag{B.3}$$

which are known to be strictly positive provided that $(g^{-1})'$ is nonzero everywhere (thus g^{-1} is strictly monotone). This is not entirely obvious, but w_i is the variance of u_i (with $a_i = 1$ and $\psi = 1$), which is nonzero whenever $(g^{-1})'$ is nonzero everywhere.

We may note that when the weights are given by (B.3), the working response z, see (9), given the linear predictor η can be computed as

$$z_i = a_i(y_i - g^{-1}(\eta_i))g'(g^{-1}(\eta_i)) + \eta_i, \tag{B.4}$$

which renders it unnecessary to compute the intermediate score statistic.

Supplementary Materials

SuppMatJCGS: SuppMatJCGS is a folder containing scripts and datasets used in the examples in Sections 4.2.1, 4.2.2, 4.3, and 6.5 along with a ReadMe file. (SuppMatJCGS.zip, zipped file).

Acknowledgments

ORCID

Niels Richard Hansen http://orcid.org/0000-0003-3883-365X

References

Bapat, R. (2010), *Graphs and Matrices (Universitext)*, New York: Springer. [716]

Beck, A., and Teboulle, M. (2009), "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems," *SIAM Journal on Imaging Sciences*, 2, 183–202. [719,721]

—— (2010), "Gradient-Based Algorithms With Applications to Signal Recovery Problems," in *Convex Optimization in Signal Processing and Communications*, eds. D. P. Palomar and Y. C. Eldar, Cambridge: Cambridge University Press, pp. 3–51. [715,716,718,721]

Bühlmann, P., and van de Geer, S. (2011), *Statistics for High-Dimensional Data: Methods, Theory and Applications* (*Springer Series in Statistics*), Berlin Heidelberg: Springer. [709,719]

Buis, P. E., and Dyksen, W. R. (1996), "Efficient Vector and Parallel Manipulation of Tensor Products," *ACM Transactions on Mathematical Software (TOMS)*, 22, 18–23. [709,710,723]

Currie, I. D., Durban, M., and Eilers, P. H. (2006), "Generalized Linear Array Models with Applications to Multidimensional Smoothing," *Journal of the Royal Statistical Society*, Series B, 68, 259–280. [709,710,712,719,722,723]

De Boor, C. (1979), "Efficient Computer Manipulation of Tensor Products," *ACM Transactions on Mathematical Software (TOMS)*, 5, 173–182. [709,723]

Eddelbuettel, D., and François, R. (2011), "Rcpp: Seamless R and C++ Integration," *Journal of Statistical Software*, 40, 1–18. [719]

Friedman, J., Hastie, T., and Tibshirani, R. (2010), "Regularization Paths for Generalized Linear Models via Coordinate Descent," *Journal of Statistical Software*, 33, 1–22. [709,712,716,720]

Hastie, T., Tibshirani, R., and Wainwright, M. (2015), *Statistical Learning With Sparsity: The Lasso and Generalizations* (*Chapman & Hall/CRC Monographs on Statistics & Applied Probability*), Boca Raton, FL: CRC Press. [721]

Horn, R. A., and Johnson, C. R. (1991), *Topics in Matrix Analysis*, Cambridge: Cambridge University Press. [716]

Lund, A. (2016), *glamlasso: Penalization in Large Scale Generalized Linear Array Models, R Package Version 2.0.1.* [709,719]

Moreau, J.-J. (1962), "Fonctions Convexes Duales et Points Proximaux dans un Espace Hilbertian," *Comptes Rendus de l'Académie des Sciences, Paris*, 255, 2897–2899. [718]

Parikh, N., and Boyd, S. (2014), "Proximal Algorithms," *Foundations and Trends® in Optimization*, 1, 127–239. [711,718,721]

Nelder, J. A., and Wedderburn, R. W. M. (1972), "Generalized Linear Models," *Journal of the Royal Statistical Society*, Series A, 135, 370–384. [710]

Roland, P. E., Hanazawa, A., Undeman, C., Eriksson, D., Tompa, T., Nakamura, H., Valentiniene, S., and Ahmed, B. (2006), "Cortical Feedback Depolarization Waves: A Mechanism of Top-Down Influence on Early Visual Areas," *Proceedings of the National Academy of Sciences*, 103, 12586–12591. [712]

Springer, M. D., and Thompson, W. E. (1970), "The Distribution of Products of Beta, Gamma and Gaussian Random Variables," *SIAM Journal on Applied Mathematics*, 18, 721–737. [719]

Tseng, P., and Yun, S. (2009), "A Coordinate Gradient Descent Method for Nonsmooth Separable Minimization," *Mathematical Programming*, 117, 387–423. [711,717]

Vincent, M., and Hansen, N. R. (2014), "Sparse Group Lasso and High Dimensional Multinomial Classification," *Computational Statistics & Data Analysis*, 71, 771–786. [716]

Wood, S. (2006), *Generalized Additive Models: An Introduction With R* (*Chapman & Hall/CRC Texts in Statistical Science*), Boca Raton, FL: CRC Press. [712]

Zhang, H., Jiang, J., and Luo, Z.-Q. (2013), "On the Linear Convergence of a Proximal Gradient Method for a Class of Nonsmooth Convex Minimization Problems," *Journal of the Operations Research Society of China*, 1, 163–186. [719]

Zou, H., and Li, R. (2008), "One-Step Sparse Estimates in Nonconcave Penalized Likelihood Models," *Annals of Statistics*, 36, 1509–1533. [719]