Spatio-temporal modeling of neuron fields

Adam Lund

PhD thesis
Department of Mathematical Sciences
University of Copenhagen

Adam Lund
adam.lund@math.ku.dk
Institut for Matematiske Fag
Københavns Universitet
Universitetsparken 5
2100 København Ø

Submitted March 31st 2017

Supervisors:
Prof. Niels Richard Hansen, IMF, University of Copenhagen
Co-Supervisor: Per Roland, INF, University of Copenhagen

Assessment committee:
Ass. Prof Anders Tolver (Chairman), University of Copenhagen
Senior Lecture Johan Lindström, Lund University
Ass. Prof. Bo Martin Bibby, Aarhus University

ISBN 9788770789349

Typeset in Hoefler Text using the LaTeX memoir-class .

Abstract

The starting point and focal point for this thesis was stochastic dynamical modelling of neuronal imaging data with the declared objective of drawing inference, within this model framework, in a large-scale (high-dimensional) data setting. Implicitly this objective entails carrying out three separate but closely connected tasks; i) probabilistic modelling, ii) statistical modeling and iii) implementation of an inferential procedure. While i) - iii) are distinct tasks that range over several quite different disciplines, they are joined by the premise that the initial objective can only be achieved if the scale of the data is taken into consideration throughout i) - iii).

The strategy in this project was, relying on a space and time continuous stochastic modelling approach, to obtain a stochastic functional differential equation on a Hilbert space. By decomposing the drift operator of this SFDE such that each component is essentially represented by a smooth function of time and space and expanding these component functions in a tensor product basis we implicitly reduce the number of model parameters. In addition, the component-wise tensor representation induce a corresponding component-wise tensor structure in the resulting statistical model. Especially, the statistical model is design matrix free and facilitates an efficient array arithmetic. Using proximal gradient based algorithms, we combine this computationally attractive statistical framework with non-differentiable regularization to form a computationally efficient inferential procedure with minimal memory foot print. As a result we are able to fit large scale image data in a mathematically sophisticated dynamical model using a relatively modest amount of computational resources in the process.

The contributions presented in this thesis are computational and methodological. The computational contribution takes the form of solution algorithms aimed at exploiting the array-tensor structure in various inferential settings. The methodological contribution takes the form of a dynamical modelling and inferential framework for spatio-temporal array data. This framework was developed with neuron field models in mind but may in turn be applied to other settings conforming to the spatio-temporal array data setup.

Resumé

Udgangspunktet og omdrejningspunktet for denne afhandling var stokastisk dynamisk modellering af neuron-billeddata med et erklæret mål om at drage inferens, i denne modelramme, for høj-dimensionalt data. Implicit indebærer dette mål tre særskilte, men alligevel tæt forbundne opgaver; i) sandsynlighedsteoretisk modellering, ii) statistiske modellering og iii) implementering af en inferensprocedure. På trods af at i) - iii) udgør særskilte opgaver, der hver især indrager vidt forskellige discipliner, deler de det præmis, at det oprindelige mål med projektet kun kan nås, hvis skalaen eller dimensionen af data, tages i betragtning i både i), ii) og iii).

Strategien i dette projekt har været, med udgangspunkt i en rum- og tidskontinuert stokastisk modelleringstilgang, at etablere en stokastisk funktional differentialligning (SFDE) på et Hilbert rum. Ved at dekomponere driftoperatoren i denne SFDE således at hver komponent er repræsenteret af en glat funktion af tid og rum og ved at udvikle disse komponentfunktioner i en tensorproduktbasis reduceres antallet af modelparametre implicit. Denne komponentvise tensorrepræsentation inducerer en tilsvarende komponentvis tensorstruktur i den afledte statistiske model. Specielt dekomponerer designmatricen hvorved man opnår en beregningsmæssig fordelagtig statistiske modelramme. En efficient inferensprocedure med et minimalt hukommelseforbrug kan så konstrueres ved at benytte proximal-gradient-baserede algoritmer til at kombinere denne statistisk modelramme med ikke-differentiabel regularisering.

Derved kan man for højdimensonalt billededata drage inferens i en matematisk set relativet sofistikeret dynamisk model med moderate beregningsmæssige midler.

Bidragene præsenteret i denne afhandling er således både beregningsmæssige og metodologiske. De beregningsmæssige bidrag har form af løsningsalgoritmer der udnytter tensorstruktur i forskellige statistiske modeller. Det metodologiske bidrag har form af en sandsynlighedteoretisk og statistisk modelleringsramme for spatio-temporal arraydata. Denne ramme blev udviklet med neuron-field modeller i tankerne, men kan i princippet anvendes i andre sammenhænge der involverer spatio-temporalt array data.

Contents

Part I

Themes

Chapter 1

Introduction

The overall focus in this thesis is on the application of mathematical (probabilistic) and statistical modelling as a means to analyze high dimensional array data with specific emphasis on neuronal brain image data. The initial grand idea was to investigate if, by specifying a model for the dynamics of brain activity in a stochastic framework, and by applying novel techniques from computational statistics, we could infer neuronal connections in the visual cortex over time and space. Especially, we were interested in extracting a type of network from data with a sparse structure corresponding to notions in the neuroscience literature regarding connectivity of neuronal networks in the brain.

In reality carrying out this idea involves, besides establishing the mathematical model and statistical method, construction and implementation of algorithms with the specific aim of drawing statistical inference in a large-scale data setting. As such the scope of this thesis extends well beyond statistics and probability theory and in reality ranges over several different subject areas including, statistics, probability theory, convex optimization, computer sciences and neuroscience and is perhaps most aptly viewed as a data scientific endeavor. Especially, quite a bit of time and energy have been allotted to computational and algorithmic aspects in this project.

The specific data motivating this endeavor was provided by PhD thesis co-advisor Prof. Per Ebbe Roland who together with his team have used a so called voltage sensitive dye technique to produce images of the visual cortex in a live animal brain being exposed to a visual stimulus. This experiment produced hundred of thousands of pictures of the visual cortex on a mesoscopic scale with very high temporal as well as spatial resolution. The aim of the experiment was to study the propagation over time and space of brain signals or brain activity resulting from the processing of this stimulus. In short, depriving the animal of every external input, except for the visual stimulus, should in principle generate a signal in the brain (or a pattern of activity) that when observed over time and space will reveal something about the structure (network) underlying brain signal propagation. That is, the idea is to gain knowledge about generic properties of brain processing of visual input on a mesoscopic scale by studying brain activity patterns observed in the data. The high spatio-temporal resolution of the images in principle makes this scientific undertaking possible and results of a purely neuroscientific analysis were reported in a series of papers, see e.g. Roland et al. (2006), Eriksson et al. (2008), Ahmed et al. (2008), Harvey et al. (2009) and Harvey and Roland (2013).

In this project we wanted to approach the neuroscientific question described above regarding neuronal connections from a statistical or data scientific angle. Using a probabilistic modelling framework to draw statistical inference for such large-scale spatio-

temporal neuronal data would in turn provide us with a principled method of inferring propagation mechanisms relevant for processing input to the brain. This however is not the conventional approach in the neuroscience literature. Especially, within computational neuroscience the focus is typically on brain data on a much smaller scale, i.e. single neurons or ensembles of single neurons. For this type of brain data, dynamical models are applied, that in an exact or deterministic way can replicate properties observed in small scale neuronal data. These dynamical models typically take the form of a system of deterministic differential equations that model precise dynamics of various known chemical or electrical reactions known to underlie the synaptic communication in the brain. For instance the classical electrophysiological models like the Hodgkin-Huxley model, FitzHugh-Nagumo model or the Morris-Lecar model of action potentials. There are many other neuronal models that characterize various aspects of neuronal dynamics and they are all based on exact electrophysiological relationships that in turn are believed to govern the neuronal communication in the brain.

Here our approach is quite different. Instead of modelling the exact dynamics in the brain based on precise electrophysiological dynamical models we want to use a more crude (but also more robust approach) that in turn can take into account natural variation as well as observation noise in the data. Compared to that of classical computational neuroscience, this approach is more suitable for data on a much larger scale that contains signals potentially emanating from billions of different cells, not all being neurons. Especially the data can contain information about the neuronal brain activity as well as various types of noise and physiological artifacts. By taking a probabilistic approach and deriving a statistical model we obtain a way to statistically infer dependence structures from the data that inherently takes noise and variation into account. As such our approach is focused on the data and its properties rather than on specific knowledge pertaining to chemical or electrical reaction mechanisms in the brain. We note that an important added benefit of this data driven approach is that the contributions made in this project are not limited to neuroscientific applications but are applicable for a wider array of problems characterized by the generic framework described in the following.

1.1 Overview

Given the introduction above we can divide the project into three parts or objectives:

i) Probabilistic modelling of spatio-temporal data in a stochastic dynamical framework.

ii) Statistical model based on i) yielding an inferential procedure.

iii) An implementation based on ii) aimed at large scale data.

In the following chapters we introduce some themes that in their own right outline the scope of the thesis and relate to i)-iii). The exposition is intended to be a survey that serves as an introduction or brief primer to the contributions in Part II and Part III and also provides a form of project narrative.

In Chapter 2 we begin by describing the data first from a generic structural point of view, i.e. the so called array structure, and then go on to describe the actual brain image data in some detail.

In Chapter 3, we introduce our approach to the modeling and analysis of the data i.e. content that relate to i) and ii). We first introduce concepts pertaining to the probabilistic modelling given by a stochastic model describing the evolution of a random field. That is, concepts and notions relating to i) above. We then discuss statistical modelling in terms

of regression modelling that in turn can be based on the probabilistic model and used to draw the inference. That is, concepts and notions relating to ii). We then introduce the structural assumptions that the statistical models in this project all share and discuss how this structure arise in our setup. Finally we finish by briefly introducing the statistical method, penalized estimation, that we use throughout to obtain sparsity.

Chapter 4 is concerned with the algorithmic and computational aspect of carrying out the statistical inferential procedure, that is iii) above and a detailed introduction to the proximal algorithm is given. Last, a type of computations or computational scheme that may be exploited in the structural framework outlined in Chapter 3 is discussed in the last section of Chapter 4 in order to motivate the algorithms proposed.

Finally, even though quite a substantial amount of resources have been spent on the actual implementation of the models and methods described below (i.e. writing software) this aspect of the project is not covered in the following chapters. However, two software implementations are distributed and freely available as R-packages and listed under contributions in Table 1.1 below.

1.2 Contributions

Table 1.1 below shows the contribution made during this project and their status as of Saturday 30th September, 2017.

Work	Journal / repository	Status
Lund (2016)	CRAN	Accepted
Lund et al. (2017)	JCGS	Accepted
Lund (2017)	CRAN	Accepted
Lund and Hansen (2017)		In preparation
Lund et al. (2017)		In preparation

Table 1.1

The R software package Lund (2016) is an implementation based on the algorithm proposed in the article Lund et al. (2017). This package solves a non-differentiable penalized estimation problem in a generalized linear array model framework for large-scale data. The package is available from the Comprehensive R Archive Network (CRAN). See Chapter 8.

In the article Lund et al. (2017) we propose a new algorithm, the gradient-descent proximal gradient (gd-pg) algorithm. We show both on simulated data and real data (brain image data and NYC taxi data) how it performs compared to the existing highly efficient procedure from Friedman et al. (2010) and discuss its convergence properties. The manuscript is accepted for publication in the Journal of Graphical and Computational Statistics. See Chapter 5.

The software package Lund (2017) is an implementation based on the algorithm proposed in the article Lund et al. (2017). This package solves the soft maximin estimation problem from Lund et al. (2017) for array-tensor structured data and is available from CRAN. See Chapter 9.

In Lund and Hansen (2017) a dynamical random field model (a stochastic functional differential equation on a Hilbert space) is constructed as a model for the spatio-temporal brain image data. Based on this model we shown how to derive a statistical model with a

certain array-tensor structure. This leads to a computationally efficient inferential procedure based on the gd-pg algorithm from Lund et al. (2017). We demonstrate how this framework can be applied to the spatio-temporal neuronal data to infer structures underlying the brain signal propagation. The manuscript is currently a preprint and is expected to be submitted within the next few months. See Chapter 6.

In Lund et al. (2017) we propose a softmaximin estimation problem for inferring common components in noisy heterogenous data. This problem can be seen as a soft version of the maximin problem from Meinshausen and Bühlmann (2015). We also propose to combine the npg-algorithm from Chen et al. (2016) with array arithmetic to solve this problem and verify the convergence of the resulting algorithm. We thereby extend the range of the gd-pg algorithm from Lund et al. (2017) to cover problems where the gradient of the loss is only locally Lipschitz. We demonstrate the approach on real heterogenous data given by the entire neuronal spatio-temporal dataset described in Section 2.2 below. The manuscript is currently a preprint and is expected to be submitted within the next few months. See Chapter 9.

Chapter 2

Data

In this chapter we will introduce and discuss the type of data considered in this project. Especially we introduce the generic structure of the data as this is instrumental in the underlying approach. We also discuss a specific example of this type of data, namely the voltage sensitive dye (VSD) recordings of brain activity in the visual cortex.

2.1 Array data structure

As discussed in the introduction the starting point, as well as the focal point, for this project was as a data set consisting of spatio-temporal recordings of the visual cortex. As such it seems natural to begin with a discussion of different aspects pertaining to this type of data. To begin with we will zoom out a little bit and introduce some basic definitions and concepts used to characterize the generic nature or structure of this type of data. Especially we will begin by defining how this type of data is organized.

To this end let $d \in \mathbb{N}$ and consider a d-dimensional lattice or regular grid denoted by $\mathcal{G}^d$ and defined as the Cartesian product

$$\mathcal{G}^d := \mathcal{X}_1 \times \cdots \times \mathcal{X}_d \tag{2.1}$$

of d discrete sets $\mathcal{X}_1, \ldots, \mathcal{X}_d$. For $i \in \{1, \ldots, d\}$, we can think of $\mathcal{X}_i = \{x_{i,1}, \ldots, x_{i,N_1}\}$, as a label set where $N_i := |\mathcal{X}_i|$ is then the number of (marginal) labels in the ith dimension. Thus we have a total of $N := \prod_i^d N_i$ d-dimensional (simultaneous) label or grid points, given by the d-tuples

$$(x_1, \ldots, x_d)_1, \ldots, (x_1, \ldots, x_d)_N. \tag{2.2}$$

A very important feature of the data considered in this thesis is that it is labeled or sampled in a d-dimensional grid like (2.1). That is we implicitly assume that a sample of data is available for each combination of marginal labels. In turn this means that the data can be organized in a so called d-dimensional array defined next.

Definition 2.1. *Let $\mathcal{A} := \{a\}$ be a set of elements and I^d the cartesian product*

$$I^d := \{1, \ldots, N_1\} \times \ldots \times \{1, \ldots, N_d\}, \quad d \in \mathbb{N},\ i \in \{1, \ldots, d\},\ N_i \in \mathbb{N}.$$

If the elements in $\mathcal{A}$ can be indexed by the set I^d we can represent $\mathcal{A}$ as $A := (a_i)_{i \in I}$ and call A a d-dimensional array *and $\mathcal{A}$ a d-dimensional array (data) set.*

By Definition 2.1 an array is simply a set that can be indexed by a regular grid I^d. We note that in the machine learning literature, what we above define as an array is typically referred to as a tensor, e.g. as in TensorFlow, Googles open source software library for machine learning. This library contains a range of machine learning tools that take arrays, as in Definition 2.1, as input.

Now suppose that to each of the N grid points in (2.2) we associate a grid value $y_{x_1,\ldots,x_d} \in \mathbb{R}$ (a data point), giving us a collection of N grid values or data points $\{y_{x_1,\ldots,x_d}\}$. Then by mapping the d-dimensional cartesian sampling grid $\mathbb{G}_d$ to the cartesian index set I^d we index the data points as in Definition 2.1 and obtain a d-dimensional array $Y := (y_i)_{i \in I^d}$ containing the data. In general we will define array data to be any kind of data that can be mapped to a d-array. We note however that this mapping may introduce distortion of the data which is for instance the case for the particular neuronal VSD data considered in the thesis as discussed in Section 2.2.4 below.

In classical statistics multiway contingency tables are examples of data that inherently have array structure. Another example from classical statistics is experiments with a factorial design. A third example that perhaps is less classical is image data with rectangular pixels. The type of data we are going to work with is image data however, while the pixels are in fact not rectangular, we will still consider this to be array data even though the array representation in principle introduce distortion in this case, see Section 2.2.4. Furthermore data that is not observed or labeled by a grid may sometimes be summarized in grid structure yielding array data. An example of this is given in Lund et al. (2017) where we consider the space-time coordinates of taxicabs in New York City. These coordinate are clearly not confined to lie in a grid however by binning the data we obtain array structured count data. In general binning data will lead to array structured data that may then be analyzed using the approaches developed in this thesis. We also note that while the definition of an array implies that all combinations of labels are observed in the data, we can in fact handle array data where there are missing observations for some label combinations.

The array characterization is obviously a quite generic way to describe a given data set, however throughout the thesis it is exactly this structure that we take in to account right from the mathematically modelling of the data to the construction of the algorithms carrying out the statistical inference in these models. Especially by exploiting this structure, as shown in Lund et al. (2017), Lund and Hansen (2017) and Lund et al. (2017), we are able to model large amount of data using sophisticated models and still obtain a computationally feasible statistical inferential procedures.

2.1.1 Spatio-temporal array data

Next let us consider a particular setting where we can encounter array data namely the so called spatio-temporal setting. In this setting we let $\mathcal{X}_i \subset \mathbb{R}$ and $x_{i,j-1} < x_{i,j}$ for all i, j. For the sake of simplicity we will assume that each $\mathcal{X}_i$ constitutes an equidistant grid in $\mathbb{R}$ with $\Delta_i = x_{i,j} - x_{i,j-1}$ for all i, j. Thus $\mathcal{G}^d \subset \mathbb{R}^d$ and we call $d-1$ dimensions, say the dimensions $1, \ldots, d-1$, the spatial dimensions and the remaining dth dimension the temporal dimension, to underline that there is a fundamental difference between them. To emphasize this difference we will denote the set of grid points in $\mathcal{X}_d$, the temporal dimension, by $\mathcal{T} := \{t_1, \ldots, t_{N_d}\}$ and refer to these points as time points and denote the spatial part of $\mathcal{G}^d$ as $\mathcal{S} := \mathcal{X}_1 \times \cdots \times \mathcal{X}_{d-1}$. We then have the spatio-temporal sampling grid

$$\mathcal{G}^d = \mathcal{S} \times \mathcal{T}.$$

As above each coordinate $(x_1, \ldots, x_d, t)$ is associated with a grid value $y_{x_1,\ldots,x_d,t} \in \mathbb{R}$ yielding $N = \prod_i^d N_i$ grid values or data points which of course may be represented using a d-dimensional array by mapping the grid $\mathcal{S} \times \mathcal{T}$ to the index set I^d. In this sense the spatio-temporal array data is just like any other array data. However, the spatio-temporal setting differs from the general array setting in the way we think of grid values and their association to the underlying grid. First, compared to the general setting we now have the extra structure of the set $\mathcal{T}$, corresponding to the direction of time. Furthermore, $\mathcal{S}$ is a subset of a $d-1$-dimensional Euclidian space and as such has a structure that can be given some geometric interpretation. Thus we can think of this spatio-temporal data as being sampled in sequential fashion as

$$\begin{array}{c} (y_{x_{1,1},\ldots,x_{d,1},t_1}, \ldots, y_{x_{1,N_1},\ldots,x_{d-1,N_{d-1}},t_1}) \\ (y_{x_{1,1},\ldots,x_{d,1},t_2}, \ldots, y_{x_{1,N_1},\ldots,x_{d-1,N_{d-1}},t_2}) \\ \vdots \\ (y_{x_{1,1},\ldots,x_{d,1},t_{N_d}}, \ldots, y_{x_{1,N_1},\ldots,x_{d-1,N_{d-1}},t_N}) \end{array} \tag{2.3}$$

yielding a collection of data points to each time point t_i having some spatial structure. For instance for $d = 3$ the structure would form 2-dimensional images and observed over time these images constitute a film. For such data, we will in the following call the collection of the grid values in each row in (2.3) for a $d-1$-dimensional frame or just a frame, to indicate that they are to be thought of simultaneously and as having some meaningful spatial structure.

We note that in general we could have array data with a temporal dimension but where the labels do not lie in Euclidian space hence have no geometric structure relating them across dimension. This type of data is usually called panel data or longitudinal data but can in principle be viewed as spatio-temporal array data where spatio then refers to label space. However, in the following when talking about spatio-temporal data we mean data observed over time with a geometrically meaningful spatial organization in Euclidian space.

2.2 Spatio-temporal neuronal data

We will here in some detail describe the data central to this project, namely the spatio-temporal voltage sensitive dye recordings of neuron activity in the visual cortex. The data set itself was recorded by Professor P. E. Roland and his team and was first described and analyzed in Roland et al. (2006). However several other papers have since been published considering the same type of data see e.g. Harvey et al. (2009), Harvey and Roland (2013) and detailed description of the data can be found there. The purpose of the experiment was to investigate the propagation of brain activity in the visual cortex resulting from the processing of visual stimulus presented to the animal while being anesthetized.

In the experiment producing the data 13 adult female ferrets were anesthetized and paralyzed and the visual cortex of each ferret was then exposed and stained for 2 hours using a voltage sensitive dye. In the experiment each ferret had one pupil dilated while the other eye was blinded. For each animal, recordings were made over 4 sessions with each session having typically 3 to 5 trials (for two animals in this data set each session had ten trials). In each trial two recordings were made; a stimulus recording and a background recording each lasting 1256 ms. In both recordings a grey background displayed on a computer screen was presented to the animal. For the stimulus recording however, after 200 ms a stationary white square, appeared on the grey background for the next 250

ms. In total the data set contains 275 trials each of which consists of two recordings. Finally, in each session, in order to measure the so called resting light intensity, that is the fluorescent light emitted in complete dark an additional recording was made during which nothing was presented to the animal.

2.2.1 Voltage sensitive dye technique

As mentioned the recordings were made using a voltage sensitive dye (VSD) imaging technique. The voltage sensitivity of the dye means that the wavelength of the light emitted from the dye changes with the membrane potential, in the dyed area. Thus this dye effectively transforms changes in the membrane potential to light intensity. Hence changes in the underlying voltage e.g. resulting from brain activity can be recorded as changes in an optical signal using a camera. Furthermore the relation between the underlying brain activity and the optical VSD signal is roughly linear implying that the VSD transformation does not enhance or diminish any parts of the voltage spectrum, see Chemla and Chavane (2010).

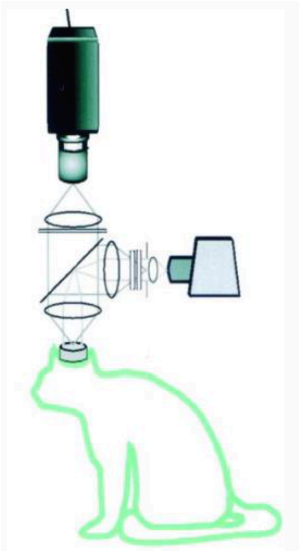


Figure 2.1: The VSD image recording setup.

In this experiment the light intensity was recorded using 464 photodiode detectors arranged in a hexagonal shape with 464 channels, see Figure 2.2, with a total diameter of 4.2 mm. In turn by using the VSD imaging technique it is possible to study brain activity in the visual cortex on a mesoscopic scale with a high spatial resolution. Using the VSD imaging technique it is also possible to sample the data with high temporal resolution as well. In this experiment the images (the hexagonal samples) were recorded with a temporal resolution equal to one sample every 0.6136 ms. A total of 2048 images were recorded for each trial over the 1256 ms the experiment lasted.

The immediate advantage of the VSD technique compared to e.g. fMRI technique is the very high spatio-temporal resolution of the data produced. In comparison the fMRI technique has a very good spatial resolution but the temporal resolution is much lower with a frequency in the orders of seconds, see Lindquist (2008). Before the advent of the VSD technique high frequency recordings of the brain were made on single neurons or on a small set of neurons only. Thus the VSD technique is applied on spatial scale that, from a neuroscience perspective, is large but without compromising the temporal resolution of the recordings.

A draw back of the VSD technique is that the data produced using this technique is easily contaminated compared to data recorded with more classical neuroscience techniques. First of all, transforming the electrical signal in to a signal made up of photons (a light signal) introduce photon noise in the recordings, see e.g. Jin et al. (2002). Secondly, the VSD technique is very sensitive, implying that the light signal is easily distorted leading to contaminated data. The concentration of dye also affects the light emitted from the dyed area. Thus if the dye is not evenly distributed and absorbed this will cause the recorded signal to be spatially distorted. Furthermore even if the dye is exactly evenly distributed the dyed areas will bleach over time causing the light intensity to decrease over time resulting in a temporal distortion of the signal.

The sensitive nature of the VSD technique has the consequence that when applied in an in vivo setting the recorded data may contain biological or physiological signals that are essentially unrelated to brain activity. These signals are typically then referred to as artifacts. One important artifact in the experimental setup described above is the so called pulse artifact. With each heartbeat the blood vessels expand thereby increasing their surface hence diluting the dye which in turn alters the emitted light. Thus the pulse will create a presumably cyclical pattern in the data that is unrelated to the brain signal propagation. In the recordings the contamination caused by the pulse artifact was in part alleviate by synchronizing the two trial recordings with the heart rate and then subtracting the two recordings, see Section 2.2.3.

On top of issues pertaining contamination the exact nature of the VSD signal is not completely obvious. Especially the signal underlying the VSD optical signal is in fact a compound signal composed of several electrical signals. First of all, compared to the more classical brain signal recording techniques the VSD is composed of signals from multiple layers of neurons. Furthermore electrical output from other types of cells than neurons (e.g. glia cells) can also affect the VSD signal. As such the VSD yields a composite signal that we can loosely call brain activity. Especially as noted in Harvey et al. (2009) the voltage sensitive dye (VSD) signal is to be treated as an *indicator* for the population membrane potential from cells in the supra-granular layers, and not the membrane potential itself.

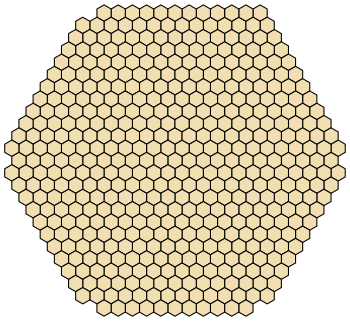


Figure 2.2: The hexagonal frame.

2.2.2 Heterogeneities across trails

In addition to the issues discussed above regarding contamination of the VSD data for single trial recordings the data will also be heterogeneous over trials. As noted above one type of heterogeneity could be caused by bleaching of the dye. However many other sources to heterogeneity can be imagined for this data. For instance given the invasive nature of the experiment the responsiveness of the animal could decline over trials caused by exhaustion. Also given that the VSD technique is quite sensitive even small changes in the experiment surroundings could also affect the recordings creating another source of heterogeneity over trials.

Physiological differences between the animals will also create heterogeneities over the trials. Especially the so called cytoarchitectural borders in the brain differs among the animals which in turn creates spatial misalignment in the data set across animals hence across trials. Finally a more endogenous animal specific type of heterogeneity would arise if the response pattern or signal somehow has an animal dependent component in the sense that the response to identical visual stimulus varies across animals even if everything else is equal.

In conclusion, whether we consider trials from one animal only or trials for several animals the data will be heterogeneous.

2.2.3 Preprocessing

Because of the different sources of contamination discussed above the raw VSD recordings are preprocessed in an attempt to clean up the data. Especially, for trial g let V_g^{stim}

denote the stimulus recording and V_g^{blank} denote the non-stimulus recording and let $V_{s(g)}^{\text{RLI}}$ denote the resting light intensity recorded in session $s(g)$ containing trail g.

In Roland et al. (2006) they first align the data by subtracting the two trial recordings i.e.

$$V_g^{\text{align}} := V_g^{\text{stim}} - V_g^{\text{blank}}.$$

This transformation has two effects. It centers the data around zero by subtracting the channel (pixel) specific level. Furthermore as noted above as both recordings are started at the same point in the heart rate cycle this transformation should alleviate the cyclical trend in the data caused by the pulse. However as the heart rate is not completely time homogeneous (can vary over time) the cyclical trend in the two recordings will differ implying that the pulse artifact may not be completely eliminated.

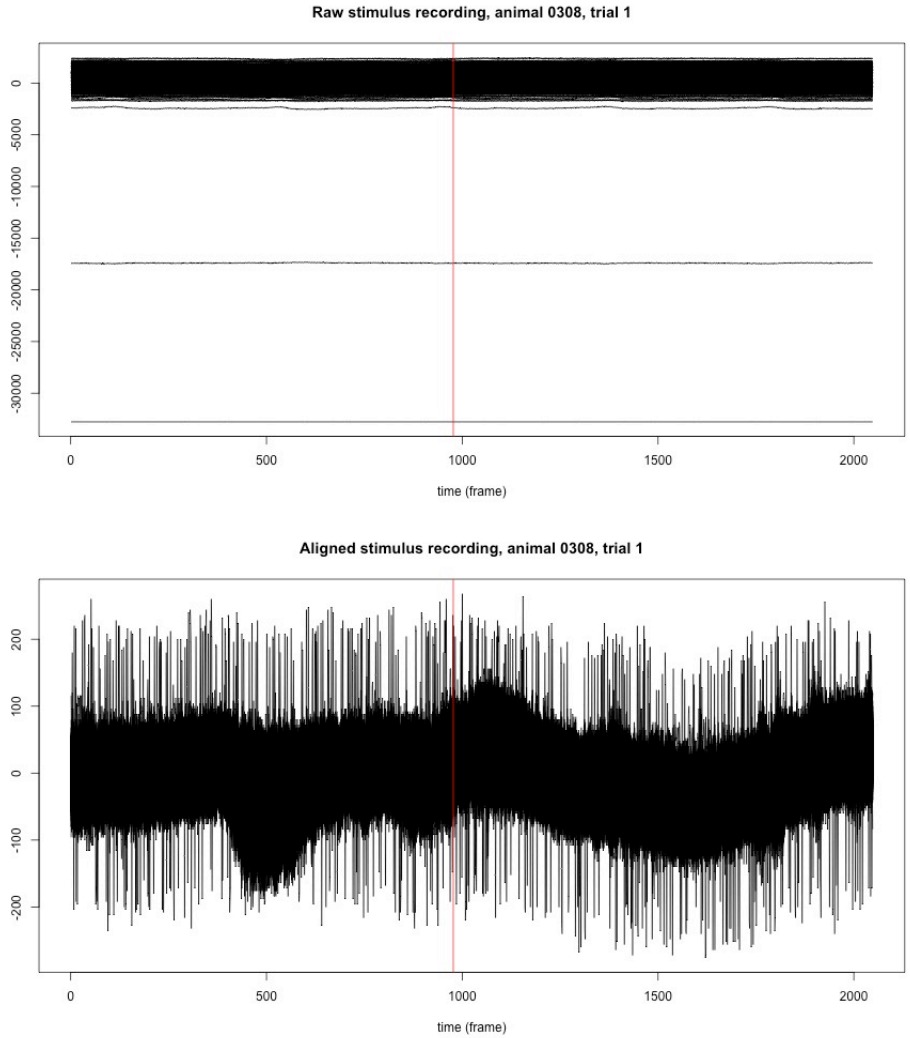


Figure 2.3: The raw data for all 464 channels for animal 0308, trial 1 (top) and the aligned (bottom). The red line indicates the cut point for this animal.

In Roland et al. (2006) they furthermore standardized the aligned data by transforming the data using the resting light intensity recording i.e.

$$\Delta V_g := \frac{V_g^{\text{align}}}{V_{s(g)}^{\text{RLI}}}.$$

The idea behind this transformation is that it will alleviate the bleaching of the dyed areas that occur over the course of the experiment. Thus by dividing by the session specific RLI the data is standardized which in turn should make the data more homogenous across trials.

We note that the data considered in this project is not standardized using the RLI recording. Instead we standardized each aligned recording V_g^{align} using the mean of the pre-stimulus part of the recording. As this mean will also decrease over time due to the bleaching this should also correct for bleaching in the data.

We note the spikes seen in the bottom display for the aligned data are caused by the data recorded in channel 249. Furthermore, the red line indicate the last time point for which data is included in any analysis. This cut off point varies from animal to animal, and for animal 0308 it is 977.

2.2.4 Array formatting

A final thing we will mention is that the VSD recordings as described above obviously do not have the array structure discussed in Section 2.1 owing to the hexagonal organization of the photo diode channels, see Figure 2.2. In order to obtain the array organization the 464 channels are mapped to a 25×25 matrix (2-array). This corresponds to first embedding the hexagonal shape inside the smallest rectangular organization of the channels covering the hexagonal shape, simply by adding "empty" channels. Then the resulting 625 channels is indexed by $\{1, \ldots, 625\}$ and this index set is then mapped, see Table 2.1, to the Cartesian index set $I^2 = \{1, \ldots, 25\} \times \{1, \ldots, 25\}$ as in Definition 2.1.

	[,1]	[, 2]	[, 3]	[, 4]	[, 5]	[, 6]	[, 7]	[,8]	[,9]	[, 10]	[,11]	[, 12]	[,13]	[, 14]	[,15]	[, 16]	[, 17]	[, 18]	[, 19]	[, 20]	[, 21]	[, 22]	[, 23]	[, 24]	[, 25]
[1,]	-	-	-	-	-	-	-	237	236	235	234	233	7	6	5	4	3	2	1	-	-	-	-	-	-
[2,]	-	-	-	-	-	244	243	242	241	240	239	238	14	13	12	11	10	9	8	-	-	-	-	-	-
[3,]	-	-	-	-	-	252	251	250	249	248	247	246	245	21	20	19	18	17	16	15	-	-	-	-	-
[4,]	-	-	-	-	260	259	258	257	256	255	254	253	29	28	27	26	25	24	23	22	-	-	-	-	-
[5,]	-	-	-	-	268	267	266	265	264	263	262	261	38	37	36	35	34	33	32	31	30	-	-	-	-
[6,]	-	-	-	277	276	275	274	273	272	271	270	269	47	46	45	44	43	42	41	40	39	-	-	-	-
[7,]	-	-	-	287	286	285	284	283	282	281	280	279	278	56	55	54	53	52	51	50	49	48	-	-	-
[8,]	-	-	297	296	295	294	293	292	291	290	289	288	66	65	64	63	62	61	60	59	58	57	-	-	-
[9,]	-	-	307	306	305	304	303	302	301	300	299	298	77	76	75	74	73	72	71	70	69	68	67	-	-
[10,]	-	318	317	316	315	314	313	312	311	310	309	308	88	87	86	85	84	83	82	81	80	79	78	-	-
[11,]	-	330	329	328	327	326	325	324	323	322	321	320	319	99	98	97	96	95	94	93	92	91	90	89	-
[12,]	342	341	340	339	338	337	336	335	334	333	332	331	111	110	109	108	107	106	105	104	103	102	101	100	-
[13,]	-	353	352	351	350	349	348	347	346	345	344	343	123	122	121	120	119	118	117	116	115	114	113	112	-
[14,]	365	364	363	362	361	360	359	358	357	356	355	354	135	134	133	132	131	130	129	128	127	126	125	124	-
[15,]	-	377	376	375	374	373	372	371	370	369	368	367	366	146	145	144	143	142	141	140	139	138	137	136	-
[16,]	-	388	387	386	385	384	383	382	381	380	379	378	157	156	155	154	153	152	151	150	149	148	147	-	-
[17,]	-	-	398	397	396	395	394	393	392	391	390	389	168	167	166	165	164	163	162	161	160	159	158	-	-
[18,]	-	-	408	407	406	405	404	403	402	401	400	399	178	177	176	175	174	173	172	171	170	169	-	-	-
[19,]	-	-	-	418	417	416	415	414	413	412	411	410	409	187	186	185	184	183	182	181	180	179	-	-	-
[20,]	-	-	-	427	426	425	424	423	422	421	420	419	196	195	194	193	192	191	190	189	188	-	-	-	-
[21,]	-	-	-	-	435	434	433	432	431	430	429	428	205	204	203	202	201	200	199	198	197	-	-	-	-
[22,]	-	-	-	-	443	442	441	440	439	438	437	436	213	212	211	210	209	208	207	206	-	-	-	-	-
[23,]	-	-	-	-	-	451	450	449	448	447	446	445	444	220	219	218	217	216	215	214	-	-	-	-	-
[24,]	-	-	-	-	-	458	457	456	455	454	453	452	227	226	225	224	223	222	221	-	-	-	-	-	-
[25,]	-	-	-	-	-	-	-	464	463	462	461	460	459	232	231	230	229	228	-	-	-	-	-	-	-

Table 2.1: Map from hexagon to 2-dimensional Cartesian index set where each "-" uniquely corresponds to a number between 465 and 625.

We note that as the original photo diode channels do not lie in a Cartesian grid this mapping will spatially distort the data in each frame. Other ways of obtaining array data from the VSD recordings are discussed in Mogensen (2016). Thus the resulting array

data is given as a 4-dimensional array with sized $25 \times 25 \times 2048 \times 275$ corresponding to 275 trial recordings of 625 channels over 2048 time steps.

We note that the resulting data array contains empty entries to all time points. If included in the statistical analysis this artificial data or lack of data can affect the analysis. This is especially relevant for the dynamical model introduced in Section 3.1 below. Consequently for this model we crop data in the spatial dimension to the largest rectangular shape containing observations of data. However, for the models considered in Lund et al. (2017) and Lund et al. (2017) we impute pre-stimulus noise in the entires with missing observations, as this does not affect the analysis and allow us to analyze all channels.

Chapter 3

Methods and modelling

In this chapter we will first briefly introduce and discuss the various concepts used in connection with modelling the neuronal data introduced above in section 2.2. We then discuss the notions and concepts related to the methods we have used to fit the models to the actual data.

3.1 A dynamical model

Given the description of the data in section 2.2 it seems natural to view the visual cortex as a random field. To introduce the notion of a random field formally let $(\Omega, \mathscr{F}, P)$ be a complete probability space endowed with an increasing and right continuous family $(\mathscr{F}_t)$ of complete sub-σ-algebras of $\mathscr{F}$. Also let $\mathcal{H}$ be suitable Hilbert space of continuous functions. From Adler and Taylor (2009) we then have the following definitions of a random field defined over a general parameter set $\mathcal{R}$ only assumed to be a topological space, however in following we shall only consider the case where $\mathcal{R} \subseteq \mathbb{R}^d$, $d \in \mathbb{N}$.

Definition 3.1. *Let $\mathcal{R}$ be a topological space and $V : \Omega \to (\mathbb{R}^{\mathcal{R}})^N$, $N \in \mathbb{N}$, a measurable map. V is called a real-valued random field when $N = 1$ and a vector-valued random field when $N > 1$. If $\mathcal{R} \subset \mathbb{R}^d$, $d \in \mathbb{N}$, we call V a (d, N) random field, and if $N = 1$, simply an d-dimensional random field.*

The term random field is typically used in situations where the geometry of the parameter set is important while the term (multi-parameter) stochastic process is used otherwise. We see that for the neuronal data discussed in section 2.2 the parameter set $\mathcal{R}$ is three dimensional with two spatial dimensions and one temporal dimension i.e. a $(3, 1)$-random field,

$$(V(r))_{r\in\mathcal{R}} := (V(x, y, t))_{(x,y,t)\in\mathbb{R}^2\times\mathbb{R}_+}.$$

To emphasize the conceptual asymmetry between the dimensions of the parameter set in this case we shall call V a *spatio-temporal random field*.

By the Kolmogorov Consistency Theorem, given mild regularity conditions, the distribution of any (d, N) random field over $\mathbb{R}^d$ is uniquely determined by its finite-dimensional distributions

$$\mathrm{P}(V(r_1) \in B_1, \ldots, V(r_n) \in B_n),$$

for all $n \in \mathbb{N}$, for all collections $\{r_1, \ldots, r_n\}$, $r_i \in \mathcal{R}$ and for all Borel sets $\{B_1, \ldots, B_n\}$, $B_i \in \mathbb{R}^N$. Using the finite dimensional distribution we can also characterize certain invariance properties of the random field V.

Definition 3.2. *Let V be an (d, N) random field defined over $\mathbb{R}^d$.*

i) V is strongly stationary if for any $\rho \in \mathbb{R}^d$

$$\mathrm{P}(V(r_1 + \rho) \in B_1, \ldots, V(r_n + \rho) \in B_n) = \mathrm{P}(V(r_1) \in B_1, \ldots, V(r_n) \in B_n),$$

for all $n \in \mathbb{N}$, collections $\{r_1, \ldots, r_n\}$, $r_i \in \mathbb{R}^d$ and Borel sets $\{B_1, \ldots, B_n\}$, $B_i \in \mathbb{R}^d$.

ii) V is strongly isotropic if for any rigid rotation R

$$\mathrm{P}(V(Rr_1) \in B_1, \ldots, V(Rr_n) \in B_n) = \mathrm{P}(V(r_1) \in B_1, \ldots, V(r_n) \in B_n),$$

for all $n \in \mathbb{N}$, collections $\{r_1, \ldots, r_n\}$, $r_i \in \mathbb{R}^d$ and Borel sets $\{B_1, \ldots, B_n\}$, $B_i \in \mathbb{R}^d$.

We can sometimes characterize the behavior of a random field using the so called mean function $m : \mathcal{R} \to \mathbb{R}$ given by

$$m(r) := E(V(r)) \tag{3.1}$$

and the covariance functions $C : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$ given by

$$C(r, r') := E(V(r) - m(r))(V(r') - m(r')). \tag{3.2}$$

Especially using (3.1) and (3.2) we can give a perhaps more concrete, and weaker, version of Definition 3.2.

Definition 3.3. *Let V be an (N, d) random field defined over $\mathbb{R}^N$.*

i) V is weakly stationary (homogeneous) if m is constant, and C is only a function of the difference $r - r'$.

ii) V is weakly isotropic if C is only a function of the Euclidean distance $\|r - r'\|$.

3.1.1 Gaussian random fields

We next introduce a fundamental family of random fields namely the Gaussian random fields, also know as the Brownian family of processes, which we shall use to model the neuronal data. Appealing to the Kolmogorov Consistency Theorem we can define the notion of a Gaussian random field.

Definition 3.4. *A Gaussian random field is a random field with all finite dimensional distributions multivariate normal.*

An especially important example of Gaussian random fields is the so called Gaussian noise which in general is an uncountable collection of independent Gaussian random variables. We will, following Adler and Taylor (2009), define these using a random measure with σ-finite control measure or intensity measure ν. With $\mathbb{B}^N$ the Borel σ-algebra

on $\mathbb{R}^N$, a Gaussian ν-noise, is a (set indexed) random field $W : \mathbb{B}^N \to \mathbb{R}$ such that for all $A, B \in \mathbb{B}^N$ with $\nu(A), \nu(B) \in \mathbb{R}$,

$$W(A) \sim N(0, \nu(A)), \tag{3.3}$$

$$A \cap B = \emptyset \Rightarrow W(A \cup B) = W(A) + W(B) \text{ a.s.}, \tag{3.4}$$

$$A \cap B = \emptyset \Rightarrow W(A) \perp W(B). \tag{3.5}$$

We note that the property in (3.5) is referred to as the independent increments property.

Having introduced the Gaussian noise we can go on and introduce the Brownian sheet also called the multi-parameter Brownian motion or Wiener process. This is done by defining an $\mathcal{R}$ indexed process $(W_r)_{r \in \mathbb{R}_+}$ using the set indexed Gaussian noise as

$$W_r := W([0, r_1] \times \cdots \times [0, r_d]), \quad r = (r_1, \ldots, r_d) \in \mathbb{R}^d_+.$$

It follows from the definition of the Gaussian noise that the covariance function for this type of random field is

$$C(r, r') = \prod_{i=1}^{d} \min(r_i, r'_i).$$

Finally for the dynamical model for the neuronal data we will need the following type of spatio-temporal Gaussian random field where we have a spatial dependence structure, see Dawson and Salehi (1980) and Peszat and Zabczyk (1997).

Definition 3.5. *Let $W(x,t)_{(x,t) \in \mathbb{R}^{d-1} \times \mathbb{R}_+}$ be a spatio-temporal random field such that*

i) $(W(x,t))_{t \in \mathbb{R}_+}$ *is an $(\mathscr{F}_t)$-Wiener process for every $x \in \mathbb{R}^{d-1}$.*

ii) The map $(\omega, x, t) \mapsto W(\omega, x, t)$ is measurable and continuous in t.

iii) $C((x,t),(x',t')) = \min\{t, t'\}\Gamma(x - x')$ *for all $t, t' \in \mathbb{R}_+$ and , $x, x' \in \mathbb{R}^{d-1}$ with Γ a function or a distribution.*

Then W is called a spatially homogeneous Wiener process.

We note that the spatially homogeneous Wiener process, cf Definition 3.3, is weakly stationary (or homogeneous) in the spatial dimension. We now proceed with giving a more concrete description of the dynamics of the random field we will use as a model for the neuronal data.

3.1.2 Stochastic functional differential equations on a Hilbert space

For fixed $t \geq 0$, $V(t)$ is a (random) bivariate function $V(t) : \mathbb{R}^2 \to \mathbb{R}$. We will for simplicity assume that this random function has a continuous version that lies in a Hilbert space $\mathcal{H}$ of bivariate continuous functions. To describe the evolution in V we will need to model the dynamics of the field. Now as we want a model that can capture the dependence over time and space we will model V using a non-Markovian type of stochastic differential equation on the Hilbert space $\mathcal{H}$, called a stochastic functional differential equation (SFDE).

To introduce this model let $\mathcal{C} := \mathcal{C}([0, \tau], \mathcal{H})$ denote a Banach space of continuous maps from the time domain $[0, \tau]$ to the Hilbert space of bivariate functions $\mathcal{H}$. Then the process $(V_t)_{t \geq 0}$ defined as

$$V_t := (V(t + s))_{s \in (-\tau, 0)},$$

is a $\mathcal{C}$-valued stochastic process defined over the parameter set $\mathbb{R}_+$. Clearly, we can think of $V_{t,}$ as the past or memory of the random field $(V(r))_{r\in\mathbb{R}_+\times\mathbb{R}^2}$ at time t.

Then let $\mu : \mathcal{C}\times[0,t] \to \mathcal{H}$ be an operator and consider a general stochastic functional differential equation on $\mathcal{H}$ given by

$$\mathrm{d}V(t) = \mu(V_t, t)\mathrm{d}t + \mathrm{d}W(t), \tag{3.6}$$

where, $(W(t))_t$ is a spatially homogeneous Gaussian random field as in Definition 3.5. We note that compared to a typical (Markovian) SDE the infinitesimal dynamic at time t is allowed to depend on the entire past from time $t-\tau$ up to time t. Hence in general a solution to (3.6), should it exist, will not be a Markov process on the Hilbert space $\mathcal{H}$.

The theory concerning general results regarding existence and stability of solutions to non-Markovian SDEs, e.g. SFDEs, on a Hilbert space is not as standard as for Markovian SDEs. In principle it should be possible to lift the SFDE (3.6) on to the Banach Space $\mathcal{C}$ where it would then be a Markovian SDE as in section 0.2 in Da Prato and Zabczyk (2014). Results for a general version of (3.6) are given in Cox (2012) where SDEs and especially SDDEs on Banach spaces are treated. Especially, Corollary 4.17 in Cox (2012) gives a existence result for a strong solution to (3.6). Also in Xu et al. (2012) a mild existence results are given for an SDDEs on a Hilbert space and for the specification introduced next this result can be strengthened to a strong solution result. In general the requirements for a solution to exist is, corresponding to the finite dimensional case, that the coefficient operators are Lipschitz continuous, which e.g. the integral operator presented next satisfies.

3.1.2.1 Neuronal model specification

The key feature of the above model is the non-Markovian property and the idea is that this path dependence should model how input to the brain is propagated over time and space. We want the drift term to model both inputs to the brain as well as model the subsequent propagation of that input. To this end it is proposed in Lund et al. (2017) to decompose the drift operator in (3.6) as

$$\mu(V_t, t) := S(t) + FV_t + HV(t). \tag{3.7}$$

Here $S : \mathbb{R}_+ \to \mathcal{H}$, $S(t) = s(x, y, t)$, $s : \mathbb{R}^3 \to \mathbb{R}$, models a deterministic exogenous input to the system and $H : \mathcal{H} \to \mathcal{H}$ $H(V(t))(x, y) := h(x, y)V(x, y, t)$ where $h \in \mathcal{H}$, a smooth function of space, captures the short range (instantaneous) memory in the system. The longer range effect on the change in the field is then in turn model by the operator F. Especially $F : \mathcal{C} \to \mathcal{H}$ is the integral operator defined by

$$F(V_t)(x, y) = \int\int_{-\tau}^{0} w(x, y, x', y', s)V(t + s, x', y')\mathrm{d}s\mathrm{d}x'\mathrm{d}y',$$

with $w : \mathbb{R}^5 \to \mathbb{R}$ a network function or convolution kernel quantifying the propagation network in the brain. With this specification we obtain a so called stochastic delay differential equation (SDDE) on the Hilbert space $\mathcal{H}$ given by

$$\begin{aligned}\mathrm{d}V(t, x, y) = s(x, y, t)\mathrm{d}t + \iint_{-\tau}^{0} w(x, y, x', y', s)V(t + s, x', y')\mathrm{d}s\mathrm{d}x'\mathrm{d}y'\mathrm{d}t \\ + h(x, y, t)V(x, y, t)\mathrm{d}t + \mathrm{d}W(t, x, y),\end{aligned} \tag{3.8}$$

with delays that are distributed over time and space (spatio-temporal distributed delays) for any non zero delay time $\tau \geq 0$. The spatio-temporal distribution of these delays is in turn controlled by the network function w which encodes how previous states of V at all (other) spatial locations (x', y') affect the (current) change in voltage at location (x, y).

Our objective is essentially, given data, to estimate or extract the drift μ. Taking the conditional expectation we get

$$\begin{aligned} \mathrm{E}(\mathrm{d}V((x,y),t) \mid \mathscr{F}_t) = s(x,y,t)\mathrm{d}t + \iint_{-\tau}^{0} w(x,y,x',y',s)V(t+s,x',y')\mathrm{d}s\mathrm{d}x'\mathrm{d}y'\mathrm{d}t \\ + h(x,y,t)V(x,y,t)\mathrm{d}t. \end{aligned} \tag{3.9}$$

The idea is then to estimate the functions appearing in the drift components i.e. w s and h in a regression model setup where the diffusion part of the model dynamics in (3.8) then plays the role of an additive error term.

3.2 Regression method

We next present the framework that we will use when estimating a conditional expectation like (3.9) given data.

To start with let us first consider two real (vector) valued random variables Y and X on some probability space $(\Omega, \mathscr{F}, \mathrm{P})$ such that the conditional expectation $\mathrm{E}(Y \mid X)$ of Y given X, is well-defined i.e. P-integrable $\mathrm{E}(|\,\mathrm{E}(Y \mid X)|) < \infty$. Next assume given a third variable B. In general B can be a random variable on Ω but typically it is assumed that B is a (deterministic) real (vector). The main idea is that B can be combined with X using a map η to somehow characterize the mean of Y conditional on X (and on B if B is random), that is

$$\mathrm{E}(Y \mid X, B) = \eta(X, B). \tag{3.10}$$

We will call the map η the regression function or predictor function, and B the regression coefficient.

In addition to this (random) parameterization of the conditional expectation it is also assumed that we have some conditional distribution of Y given X and B specified in terms of a known density function

$$y \mapsto f_{Y|X,B}(y).$$

Together the equation (3.10) and the density $f_{Y|X,B}$ determine a regression model. We usually call Y the response or the output, X the covariate, the input or the predictor and B the regression parameter.

3.2.1 The linear model

In order to make the regression model operational we will have to place assumptions on the two components, the map η and the density $f_{Y|X,B}$. Here the most crucial assumption is on the map η. In our framework the map η i.e. the conditional expectation of Y given X and B, is linear in the sense that

$$g(\eta(X, B)) = X^\top B \tag{3.11}$$

where $g : \mathbb{R} \to \mathbb{R}$ is a monotonic function.

The purpose of the regression model is given data $(x_1, y_1), \ldots, (x_1, y_n)$ of (X, Y) to use this evidence to learn η. This procedure is usually called regression of Y on X.

We will next continue by discussing three special cases of this framework, that we consider in this project where the conditional expectation is assumed linear in a set of (possibly random) parameters as in (3.11).

3.2.1.1 Generalized linear model

First assume that $B = \theta$ is real vector valued and that $(Y_1, X_1), \ldots, (Y_n, X_n)$ are conditionally independent. Then assuming that the probability distribution for $Y \mid X$ is a member of the exponential family i.e given by the two-parameter family of densities

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

gives us a generalized linear model (GLM). 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 w.r.t. some reference measure.

In this setup the negative likelihood or the loss is given as

$$l(\theta) := \sum_{i=1}^{n} a_i(y_i \vartheta(\eta_i(\theta)) - b(\vartheta(\eta_i(\theta)))) \tag{3.12}$$

which is differentiable and convex. Especially we can either obtain the MLE analytically in closed form or otherwise by using a numerical procedure as described in chapter 4 below. We consider this model setup in Lund et al. (2017) for a class of models where the covariates have a special tensor structure.

3.2.1.2 Soft maximin effects model

Next considered the setup in Meinshausen and Bühlmann (2015). Here a linear model for n response variables $Y_1, \ldots, Y_n$ and n i.i.d. covariates $X_1, \ldots, X_n$ is given as

$$Y_i = X_i^\top B_i + \varepsilon_i.$$

where $B_1, \ldots, B_n$ are random variables. We also assume that X_i and B_i are independent and that ε_i is a zero-mean real random variable uncorrelated with X_i. Neither $B_1, \ldots, B_n$ nor the error terms $\varepsilon_1, \ldots, \varepsilon_n$ need to be i.i.d. We assume that the n observations are organized in G known groups with the regression variable fixed within each group i.e. $B_g = b_g$ for group g. For the gth group let Y_g denote the response vector, X_g denote the covariate matrix (design matrix). Then for some real vector θ the empirical explained variance in group g obtained for this θ is

$$\hat{V}_g(\theta) := \frac{1}{n_g}(2\theta^\top X_g^\top y_g - \theta^\top X_g^\top X_g \theta).$$

In Lund et al. (2017) we then propose maximizing a soft minimum of the explained variance which in turn is done by minimizing the loss function

$$l_\zeta(\theta) := \frac{\log(\sum_g e^{-\zeta \hat{V}_g(\theta)})}{\zeta}. \tag{3.13}$$

We consider this loss for the neuronal data where we group the observations according to trials (see the discussion of the neuronal data in section 2.2). The loss (3.13) is a soft version of the maximin loss from Meinshausen and Bühlmann (2015).

3.2.1.3 Linear model with correlated normal errors

In a third model considered in Lund and Hansen (2017) the regression parameter $B = \theta$ is real (deterministic) vector and $\varepsilon_1, \ldots, \varepsilon_n$ are normal but not necessarily i.i.d.. This leads to a linear model we can write as

$$Y_i = X_i^\top \theta + \varepsilon_i.$$

In Lund and Hansen (2017) we consider this model where the errors have block diagonal covariance matrix with blocks given by a matrix Σ. With Y the response vector and X the design matrix we then consider the following loss function

$$l(\theta, \Sigma) := \frac{M}{2} \log|\Sigma| + \|I_M \otimes \Sigma^{-1/2}(Y - X\theta)\|_2^2. \tag{3.14}$$

which we propose minimizing using the approximate MCRE algorithm from Rothman et al. (2010) in order to fit the dynamical model discussed in section 3.1.2 to data.

3.2.2 Multi component array tensor strucuture

The examples above are types of various statistical models considered in this project. The important common feature is the linearity of the representation of the conditional expectation in the regression parameters. This implies that the conditional expectation can be computed essentially as a (covariate) matrix-vector product. In this project partly owing to the array structure of the data and partly owing to the model specification we obtain regression models with a specific covariate structure, so-called tensor structured covariates. In the following we will refer to this structure as the multi component array tensor (MCAT) framework or structure as outlined next. We note that this framework, or set of structural assumptions, is considered in Lund et al. (2017) as a multi component version of the structural assumptions considered in Currie et al. (2006). We also note that MCAT is only a set of structural assumptions placed on a model hence no distributional assumptions are made below.

First we define a trivial extension of the vec operator to d-arrays. Consider a $r \times c$ matrix (2-array) as a tuple of length c of 1-arrays of size r,

$$\begin{bmatrix} a_{11} & \ldots & a_{1c} \\ \vdots & & \\ a_{r1} & \ldots & a_{rc} \end{bmatrix} = \left[\begin{bmatrix} a_{11} \\ \vdots \\ a_{r1} \end{bmatrix}, \quad \cdots \quad , \begin{bmatrix} a_{1c} \\ \vdots \\ a_{rc} \end{bmatrix} \right].$$

We can say we have 1 nested tuple of 1-arrays of size r. Similarly, consider a $r \times c \times s$ cube (3-array) as a tuple of length s of 2-arrays of size $r \times c$, hence, as tuple of length s of tuples of length c of 1-arrays of size r,

$$\begin{aligned} &\left[\begin{bmatrix} a_{111} & \ldots & a_{1c1} \\ \vdots & & \\ a_{r11} & \ldots & a_{rc1} \end{bmatrix}, \quad \cdots \quad , \begin{bmatrix} a_{11s} & \ldots & a_{1cs} \\ \vdots & & \\ a_{r1s} & \ldots & a_{rcs} \end{bmatrix} \right] \\ &= \left[\left[\begin{bmatrix} a_{111} \\ \vdots \\ a_{r11} \end{bmatrix}, \quad \cdots \quad , \begin{bmatrix} a_{1c1} \\ \vdots \\ a_{rc1} \end{bmatrix} \right], \quad \cdots \quad , \left[\begin{bmatrix} a_{11s} \\ \vdots \\ a_{r1s} \end{bmatrix}, \quad \cdots \quad , \begin{bmatrix} a_{1cs} \\ \vdots \\ a_{rcs} \end{bmatrix} \right] \right]. \end{aligned}$$

We say we have 2 nested tuples of 1-arrays. Continuing this way we can in general view a d-array as $d-1$ nested tuples of 1-arrays of size r.

Definition 3.6. *For any d-array the* vec ***operator*** *concatenates the $d-1$ nested tuples of 1-arrays by the column dimension yielding one vector.*

In the MCAT framework the response vector y is then given as

$$y := \text{vec}(Y),$$

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

$$X := [X_1 \mid X_2 \mid \ldots \mid X_c],$$

where the rth component is a tensor product,

$$X_r = X_{r,d} \otimes X_{r,d-1} \otimes \cdots \otimes X_{r,1}, \tag{3.15}$$

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

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

This data structure induces a corresponding structure on the parameter vector, θ, which is given as

$$\theta := \begin{bmatrix} \text{vec}(\Theta_1) \\ \vdots \\ \text{vec}(\Theta_c) \end{bmatrix}$$

with Θ_r a $p_{r,1} \times \cdots \times p_{r,d}$ d-dimensional array.

We note that for a large-scale data settings the design matrix X is often too large to store in the memory of a typical computer and in general will be difficult to handle. However we also note that the design matrix X is completely characterized by the marginal design matrices. This property is exploited throughout in this project and implies that we may obtain computationally efficient routines for solving the estimation problems that only rely on these marginal design matrices.

Next we will describe one setting where this structural framework arise.

3.2.3 The tensor product basis functions

Now in order to draw inference in the dynamical model framework from subsection 3.1.2.1 we will need to parameterize the conditional expectation (3.9) so it becomes linear in a set of parameters as in the linear models in subsection 3.2.1. Considering the conditional expectation in (3.9) we see that it is given as a decomposition with each component represented by a presumably smooth multivariate function i.e. s, w and h respectively. In order to estimate the conditional expectation we will parameterize these functions. This is achieved by exploiting the fact that for any function $f : \mathbb{R}^d \to \mathbb{R}$ lying in a Hilbert space we can write

$$f(x_1, \ldots, x_d) = \sum_{k_1,\ldots,k_d} \beta_{k_1,\ldots,k_d} \phi_{k_1,\ldots,k_d}(x_1, \ldots, x_d) \tag{3.16}$$

where $(\beta_{k_1,\ldots,k_d})_{k_1,\ldots,k_d}$ is a $\prod_i k_i$ dimensional vector of coefficients and $\phi_{k_1,\ldots,k_d}$ a set of basis functions. Then by specifying the set of basis functions the idea is to infer the basis parameters using the linear regression model from subsection 3.2.1.

A thing to consider when specifying a basis of d-variate functions is the possible asymmetry between the d dimensions. For instance for $d = 3$ one dimension could represent time while the other two represent 2-dimensional space. In this situation we would expect the signal to behave differently in the time dimension compared to the space dimension e.g. posses different degrees of smoothness. Ideally we would like the basis functions $(\phi_i)_i$ to capture this fundamental difference or asymmetry between the dimensions. Furthermore we may want the basis functions to posses other mathematical properties e.g. they should be orthogonal or may be we would like them to be localized, meaning that they have compact support. Such properties can be easy to obtain when working in one dimension but there may not be any obvious recipe on how to obtain these properties in several dimensions. For instance, as noted in Meyer (1995) the recipe that works in one dimension to construct wavelet basis functions with compact support does not work in several dimensions. Thus if we want to use wavelets with compact support in d dimensions the only solution seems to be to resort to d-variate functions that are separable across the dimensions (see p. 108 in Meyer (1995)).

To explain the tensor terminology a little bit, let $d = 2$ and let $\mathcal{V}, \mathcal{W}$ be two vector spaces and consider the Cartesian product space $\mathcal{V} \times \mathcal{W}$ which is the space of all linear combinations of pairs (v, w) with $v \in \mathcal{V}$ and $w \in \mathcal{W}$. Then it is a fact that there exists a vector space, usually denoted $\mathcal{V} \otimes \mathcal{W}$, equipped with a bilinear map (usually denoted $\otimes$ correspondingly), $\otimes : \mathcal{V} \times \mathcal{W} \to \mathcal{V} \otimes \mathcal{W}$ which is unique up to an isomorphism. Together the pair $(\otimes, \mathcal{V} \otimes \mathcal{W})$ constitutes a tensor product construction of $\mathcal{V}$ and $\mathcal{W}$ and $\mathcal{V} \otimes \mathcal{W}$ is the tensor product space. By definition $\mathcal{V} \otimes \mathcal{W}$ is a quotient space of the free vector space of $F(\mathcal{V} \times \mathcal{W})$ and the subspace N of $F(\mathcal{V} \times \mathcal{W})$ consisting of the elements in $F(\mathcal{V} \times \mathcal{W})$ that are bilinearly related. The key property of this construction is that the bilinear map $\otimes$ is generic or universal in the sense that for a vector space $\mathcal{X}$, any bilinear map, $b : \mathcal{V} \times \mathcal{W} \to \mathcal{X}$, can be written as the composition $b = l \circ \otimes$ where l is a unique linear map $l : \mathcal{V} \otimes \mathcal{W} \to \mathcal{X}$.

Then to make the connection to our setup consider a Hilbert spaces $\mathcal{H}(E)$ of univariate functions on $E \subseteq \mathbb{R}$. It follows that with $(\phi_i)_i$ a basis for $\mathcal{H}(E)$ then $(\phi_i \otimes \phi_j)_{i,j}$ is a basis for the (Hilbert) tensor product space $\mathcal{H}(E) \otimes \mathcal{H}(E)$ of bivariate functions and the set $(\phi_i \phi_j)_{i,j}$ is a is a basis for the Hilbert space $\mathcal{H}(E \times E)$, see Reed and Simon (1972). Then the map l defined on $(\phi_i \otimes \phi_j)_{i,j}$ such that $\phi_i \otimes \phi_j \mapsto_l \phi_i \phi_j$ maps a basis for $\mathcal{H}(E) \otimes \mathcal{H}(E)$ to a basis for $\mathcal{H}(E \times E)$. Extending l linearly to a surjective map we have for any $f, g \in \mathcal{H}(E)$ where $f = \sum_{k_1} a_{k_1} \phi_{1,k_1}$ and $g = \sum_{k_2} a_{k_2} \phi_{2,k_2}$, using the bilinearity of $\otimes$ and the linearity of l, that

$$l(f \otimes g) = l\Big(\sum_{k_1} \sum_{k_2} a_{k_1} a_{k_2} (\phi_{1,k_1} \otimes \phi_{2,k_2}) \Big) = \sum_{k_1} \sum_{k_2} a_{k_1} a_{k_2} \phi_{1,k_1} \phi_{2,k_2} = fg.$$

This shows that $\mathcal{H}(E \times E)$ and $\mathcal{H}(E) \otimes \mathcal{H}(E)$ are isomorphic via the map l and we may simply think of $\mathcal{H}(E \times E)$ as tensor product space of bivariate functions thus use linear combinations of tensors products to represent any element in $f \in \mathcal{H}(E \times E)$.

Especially, for any $d \in \mathbb{N}$ we consider d marginal or univariate sets of basis functions for $\mathcal{H}(E)$

$$(\phi_{1,k_1})_{k_1=1}^{p_1}, \ldots, (\phi_{d,k_d})_{k_d=1}^{p_d}, \tag{3.17}$$

where $\phi_{j,m} : \mathbb{R} \to \mathbb{R}$ for $j = 1, \ldots, d$ and $m = 1, \ldots, p_j$. Then, by the tensor product construction we can specify the d-variate basis functions in (3.16) in terms of d (marginal)

sets of the univariate functions given by

$$\phi_{m_1,\ldots,m_d} \coloneqq \phi_{1,m_1} \otimes \phi_{2,m_2} \otimes \cdots \otimes \phi_{d,m_d}. \tag{3.18}$$

Evaluating each of the p_j univariate functions in the n_j points in $\mathcal{X}_j$ from subsection 2.2.1 results in an $n_j \times p_j$ matrix $\Phi_j = (\phi_{j,m}(x_k))_{k,m}$. It then follows that with Φ the $n \times p$ ($p := \prod_{j=1}^{d} p_j$) tensor product (or Kronecker) matrix

$$\Phi = \Phi_d \otimes \cdots \otimes \Phi_1 \tag{3.19}$$

we can evaluate f in (3.16) in each point in the grid $\mathcal{X}_1 \times \ldots \times \mathcal{X}_d$ as

$$f(x_1, \ldots, x_d) = (\Phi\beta)_{j_1,\ldots,j_d}.$$

Especially by reorganizing the coefficient vector β as an array we obtain the MCAT structure introduced above.

It turns out, as shown in Lund and Hansen (2017), that using this tensor product construction the conditional expectation in the dynamical model from subsection 3.1.2.1 can be framed as an MCAT structured linear model with three components corresponding to each of the components in the drift in (3.7). Furthermore in Lund et al. (2017) and Lund et al. (2017) we consider models where the conditional expectation can be considered as a special case of (3.9) where we only have one component namely the 3-variate function s corresponding to an MCAT structured model with one component. In Lund et al. (2017) this type of model is fitted to data using the GLM framework from subsection 3.2.1.1 and in Lund et al. (2017) using the maximin effects approach from subsection 3.2.1.2.

3.2.4 Penalized regression problems

We end this chapter by introducing an estimation technique or perhaps an estimation approach used in computational statistics and signal processing. For high-dimensional problems where the number of parameters is large it can be convenient and sometimes necessary to control the parameters or features used in the solution to make the problem well-posed. This can for instance happen if we use a so called overcomplete basis, a basis where the elements are not linearly independent, to represent the function f in (3.16). In this case we will need to restrict the number of basis functions we use to represent f. One way of doing this is by way of a subset or feature selection method where only basis functions that are somehow deemed important in representing f are used.

Another way of controlling the parameter space to obtain a well-posed problem is by regularization or penalization. Consider the models in subsections 3.2.1.1 - 3.2.1.3. In order to estimate the regression parameters in theses models respectively we need to minimize this loss function given by related loss function l in (3.12)-(3.14) respectively. Here in this project we minimize regularized or penalized versions of these loss functions. That is, we consider the following constrained optimization problem

$$\min_{\theta} l(\theta) \quad \text{subject to} \quad J(\theta) \leq t \tag{3.20}$$

where $t \in (0, \infty]$, $\theta \in \mathbb{R}^p$, and J is a convex penalty function. If l is the log-likelihood then with $t = \infty$ (3.20) is the usual (unconstrained) maximum likelihood problem while for $t < \infty$ (3.20) is a constrained or penalized maximum likelihood problem.

The problem (3.20) has an equivalent Lagrange formulation given by

$$\min_{\theta}\{l(\theta) + \lambda J(\theta)\} \tag{3.21}$$

where $\lambda \geq 0$ is the penalty or regularization parameter controlling the amount of penalization.

Solving (3.21) will, depending on the properties of the penalty function J, restrict the resulting parameter estimates compared to the unpenalized solution when $\lambda = 0$. The most well known form of regularization is Tikhonov regularization or ridge regression where $J(\theta) = \|\theta\|_2^2$ - the ℓ_2-penalty. With this penalty solution with small ℓ_2-norm is chosen leading to so called shrinkage. In general regularizing the estimation problem prevents so called over fitting leading to a better prediction accuracy or out of sample performance. When estimating a function f i.e. when regression coefficient θ is the vector of basis coefficients, this implies that the estimate of f for larger values of λ becomes more smooth.

Another popular choice for regularization is the ℓ_1-penalty given by $J(\theta) := \|\beta\|_1$. This choice of penalty induce sparsity in the estimate of θ essentially because there is an edge in the feasible set when ever $\theta_i = 0$. This especially implies that, in addition to regularizing the problem, the ℓ_1-penalty also performs subset selection as mentioned above.

Chapter 4

Algorithms and computing

In this chapter we will discuss the generic versions of algorithms that we have used to solve the problem (3.21) for large-scale models with the tensor-array structure introduced in subsection 3.2.2. To understand the algorithm and its convergence properties we will need some concepts from convex analysis and some results for set-valued operators related to convex functions.

We finish by discussing the so called array arithmetic which is a way to efficiently carry out matrix-vector product for tensor structured matrices which provides the sole justification for formulating the multi component tensor array model framework as discussed section 3.2.2.

4.1 Convex functions and setvalued operators

In the following let $\mathcal{H}$ be a Hilbert space equipped with the usual norm, $\|\cdot\|_2$, Let us start with the following basic definitions.

Definition 4.1. *Let $g : \mathcal{H} \to \mathcal{H}$. We say the function g is:*

i) Convex if for $\alpha \in [0, 1]$

$$g(\alpha x - (1-\alpha)y) \leq \alpha g(x) - (1-\alpha)g(y) \tag{4.1}$$

ii) Strictly convex if (4.1) *is a strict inequality.*

iii) Stongly convex if $g(x) - \|x\|_2^2$ is convex.

We say a convex function is proper if it takes values in the extended real numbers $(-\infty, +\infty]$. We also note that if g in definition 4.1 is twice differentiable the strong convexity is equivalent to

$$\nabla^2 g(x) - \nu I \quad \text{positive definite},$$

where $\nu > 0$ and $\nabla^2 g$ is the Hessian of g.

Then let $f : \mathcal{H} \to \mathbb{R}$ be a convex differentiable function with L_f-Lipschitz continuous gradient for some $L_f > 0$ and let $g : \mathcal{H} \to (-\infty, +\infty]$ be a proper convex function and consider a generic version of the unconstrained optimization problem (3.21) in section 3.2.4 given by

$$\min_x \{f(x) + g(x)\}. \tag{4.2}$$

Now as g is not necessarily differentiable we cannot characterize the solution to (4.2) simply by differentiating the objective $F := f + g$. Instead we need the so called sub-differential operator to characterize solutions to (4.2). The sub-differential of f denoted $\partial g : \mathcal{H} \to \mathcal{H}$ is defined to be the set of all sub-gradients of g in x i.e.

$$\partial g(x) := \{y \in \mathcal{H} : g(z) \geq g(x) + \langle z - x, y\rangle, \forall z \in \operatorname{dom} g\}. \tag{4.3}$$

If g is differentiable in x then clearly $\partial g(x)$ is equal to the gradient $\nabla g(x)$. However in general ∂g is a multi-valued or set-valued operator. We next define some properties pertaining a general set-valued operator $T : \mathcal{H} \to \mathcal{H}$, that we will need in order to understand the algorithm we will use to solve (4.2).

Definition 4.2. *Let $T : \mathcal{H} \to \mathcal{H}$ be a set-valued operator.*

i) *T is L-**Lipschitz continuous** if*

$$\|Tx - Ty\| \leq L\|x - y\|, \quad \forall x, y \in \mathcal{H}.$$

ii) *T is called **non expansive** if it is 1-Lipschitz.*

iii) *T is called **firmly non expansive** if*

$$\|Tx - Ty\|^2 \leq \langle Tx - Ty, x - y\rangle, \quad \forall x, y \in \mathcal{H}.$$

iv) *T is called α-**averaged** if*

$$T = (1 - \alpha)I + \alpha N,$$

where N is a non expansive operator, I the identity and $\alpha \in [0, 1)$.

v) *T is called **monotone** if*

$$\langle x - x', y - y, \rangle \geq 0, \quad \textit{for any} \quad y \in Tx,\ y' \in Tx',$$

*and **maximal monotone** if in addition the graph of T,*

$$\{(x, y) \in \mathcal{H} \times \mathcal{H} : y \in Tx\},$$

is not contained in the graph of any other monotone operator $T' : \mathcal{H} \to \mathcal{H}$.

vi) *T is γ-**inverse strongly monotone** if there exists $\gamma > 0$ such that*

$$\operatorname{Re}(\langle Tx - Ty, x - y\rangle) \geq \gamma \|Tx - Ty\|_2^2.$$

Given these definitions we first note the the sub-differential operator is in fact monotone. Especially, let $y \in \partial g(x), y' \in \partial g(x')$ then by (4.3)

$$g(z) \geq g(x) + \langle z - x, \partial g(x)\rangle \quad \text{and} \quad g(z) \geq g(x') + \langle z - x', \partial g(x')\rangle$$

for all $z \in \mathcal{H}$ especially

$$\begin{aligned} & g(x') + g(x) \geq g(x) + \langle x' - x, y\rangle + g(x') + \langle x - x', y'\rangle \\ \Leftrightarrow \\ & \langle x' - x, y' - y\rangle \geq 0 \end{aligned}$$

The next two results show why the definition of an averaged operators is useful to work with.

Theorem 4.3. *The composition of averaged operators is an averaged operator.*

Proof. Let $T = (1-\alpha)I + \alpha N$ and $S = (1-\beta)I + \beta M$ be two averaged operators. Obviously if $\alpha = 0$ or $\beta = 0$ then the composition is averaged since either $TS = S$ or $TS = T$. So assume that $\alpha, \beta \neq 0$, define $\gamma := \beta + \alpha(1-\beta)$ and note that $0 < \gamma < 1$ as $\gamma = 1 - (1-\alpha)(1-\beta)$. Then

$$TS = (1-\alpha)(1-\beta)I + \alpha(1-\beta)N + (1-\alpha)\beta M + \alpha\beta NM = (1-\gamma)I + \gamma K$$

with $K := (\alpha(1-\beta)N + (1-\alpha)\beta M + \alpha\beta NM)/\gamma$. Now using the triangle inequality and that N and M hence NM are non expansive we get

$$\begin{aligned}\|Kx - Ky\| &\leq \frac{1}{\gamma}(\alpha(1-\beta)\|N(x-y)\| + (1-\alpha)\beta\|M(x-y)\| + \alpha\beta\|NM(x-y)\|) \\ &\leq \frac{1}{\gamma}(\alpha(1-\beta)\|x-y\| + (1-\alpha)\beta\|x-y\| + \alpha\beta\|x-y\|) \\ &= \|x-y\|\end{aligned}$$

showing that K is non expansive hence TS is averaged. □

Now suppose that $T = (1-\alpha)+\alpha N$ is an averaged operator an consider the sequence produced by iteratively applying T to some $x \in \operatorname{dom} T$ i.e.

$$x_n := T^n x = TT^{n-1}x = (1-\alpha)x_{n-1} + \alpha N x_{n-1}.$$

This type of iterates are called Krasnoselskii-Mann iterates after Mann (1953) and Krasnoselsky (1955) and they always converge to a fixed point of T given that one exists.

Theorem 4.4. *Let $T = (1-\alpha)+\alpha N$ denote any averaged operator. Then $(x_n)_n := (T^n x)_n$ for some $x \in \operatorname{dom} T$ converge to a fixed point of T if any such exists.*

Proof. Note that if $T^n x \to z$ and $Tz = y$ then since T is non expansive

$$\|T^n x - y\| \leq \|T^{n-1}x - z\| \to 0$$

i.e. $T^n x \to y$ implying $y = z$ hence z is a fixed point. Further let $S := I - T$ then by the polarization identity

$$\begin{aligned}\|x_n - z\|^2 - \|x_{n+1} - z\|^2 &= \|x_n - z\|^2 - \|Tx_n - Tz\|^2 \\ &= 2\operatorname{Re}(\langle Sx_n - Sz, x_n - z\rangle) - \|Sx_n - Sz\|^2.\end{aligned}$$

Furthermore $I - N$ is $1/2$-inverse strongly monotone and the complement $S = \alpha(I-N)$ is $1/(2\alpha)$-inverse strongly monotone hence

$$2\operatorname{Re}(\langle Sx_n - Sz, x_n - z\rangle) \geq 2 \cdot \frac{1}{2\alpha}\|Sx_n - Sz\|^2 > \|Sx_n - Sz\|^2$$

as $\alpha \in (0,1)$. So $(\|x_n - z\|^2)_n$ is strictly decreasing and bounded below by 0 hence must converge. This implies that $(x_n)_n$ has a limit point x^* which must be a fixed point of T. □

We need the following lemma.

Lemma 4.5. *Consider the operator $T : \mathcal{H} \to \mathcal{H}$ and $I : \mathcal{H} \to \mathcal{H}$ the identity.*

i) T is firmly non expansive if and only if it is $1/2$-averaged.

ii) T is firmly non expansive if and only if $I - T$ is firmly non expansive.

Proof. (i): Consider the operator $(I+N)/2$ for some operator N and I the identity. Then

$$\left\langle \frac{1}{2}(I+N)x - \frac{1}{2}(I+N)y, x-y \right\rangle = \frac{1}{2}\|x-y\|^2 + \frac{1}{2}\langle Nx - Ny, x-y\rangle$$

and

$$\left\| \frac{1}{2}(I+N)x - \frac{1}{2}(I+N)y \right\|^2 = \frac{1}{4}\|x-y\|^2 + \frac{1}{4}\|Nx-Ny\|^2 + \frac{1}{2}\langle Nx-Ny, x-y\rangle.$$

Subtracting these two equation gives us the identity

$$\begin{aligned}\left\langle \frac{1}{2}(I+N)x - \frac{1}{2}(I+N)y, x-y \right\rangle - \left\| \frac{1}{2}(I+N)x - \frac{1}{2}(I+N)y \right\|^2 \\ = \frac{1}{4}(\|x-y\|^2 - \|Nx-Ny\|^2).\end{aligned}$$

If the operator $(I+N)/2$ is $1/2$-averaged N is non expansive and the right hand side is positive making the operator $(I+N)/2$ firmly non expansive.

If, on the other hand, the operator $(I+N)/2$ is firmly non expansive the left hand side is positive implying N is non expansive hence $(I+N)/2$ is $1/2$-averaged.

(ii): If T is firmly non expansive then by i) $T = (I+N)/2$ for some non expansive operator N. Then as $I - T = (I-N)/2$ and as $-N$ is also non expansive it follows from i) that $I - T$ must be firmly non expansive. □

Now suppose that x^* solves the problem (4.2). Then

$$\begin{aligned}0 \in \nabla f(x^*) + \partial g(x^*) &\Leftrightarrow 0 \in -\delta\nabla f(x^*) - \delta\partial g(x^*) \\ &\Leftrightarrow x^* + \delta\partial g(x^*) \in x^* - \delta\nabla f(x^*) \\ &\Leftrightarrow x^* \in (I+\delta\partial g)^{-1}(I-\delta\nabla f)x^*\end{aligned}$$

Thus the solution set to (4.2) is the set of fixed points for the operator

$$(I+\delta\partial g)^{-1}(I-\delta\nabla f) : \mathcal{H} \to \mathcal{H}. \tag{4.4}$$

This operator is known as the *forward-backward operator* and as we will show next for f convex with Lipschitz continuous gradient, this operator is exactly an averaged operator. Hence by the results above above we can find the solution x^* to (4.2) simply by iteratively applying the forward backward operator to a point in its domain.

We first we need the following result from also known as the descent lemma (see Bertsekas (1999)).

Lemma 4.6 (Descent lemma). *Let $f : \mathcal{H} \to \mathbb{R}$ be a differentiable convex function with L-Lipschitz gradient $\nabla f : \mathcal{H} \to \mathcal{H}$. Then*

$$f(x+y) \leq f(x) + \langle y, \nabla f(x)\rangle + \frac{L}{2}\|y\|^2.$$

The next result is key to establishing the convergence of the proximal algorithm.

Theorem 4.7. *Let $f : \mathcal{H} \to \mathbb{R}$ be a differentiable convex function with non expansive gradient $\nabla f : \mathcal{H} \to \mathcal{H}$. Then ∇f is firmly non expansive.*

Proof. Let $h : \mathcal{H} \to \mathbb{R}$ be differentiable and convex with non-expansive gradient. Then for any $t \in \mathcal{H}$ from Lemma 4.6 above

$$h(t - \nabla h(t)) \leq h(t) + \langle -\nabla h(t), \nabla h(t) \rangle + \frac{1}{2}\| - \nabla h(t)\|^2 = h(t) - \frac{1}{2}\|\nabla h(t)\|^2.$$

Using this for any $s \in \mathcal{H}$ we can get

$$\inf_t h(t) = \inf_t h(t - \nabla f(t)) \leq \inf_t h(t) - \frac{1}{2}\|\nabla h(t)\|^2 \leq h(s) - \frac{1}{2}\|\nabla h(s)\|^2. \tag{4.5}$$

Next fix $x, y \in \mathcal{H}$ and define for any fixed $s \in \mathcal{H}$ the function

$$g_s(t) := f(t) - f(s) + \langle \nabla f(s), s - t, \rangle.$$

Then g_x is convex since f is convex, $t \mapsto x - t$ is convex, and the inner product is bilinear. Furthermore $\nabla g_x(t) = \nabla f(t) - \nabla f(x)$, which is non expansive, as ∇f is non expansive, and equal to zero exactly when $t = x$ hence $\inf_t g_x(t) = 0$. Now using (4.5) on g_x we get

$$\begin{aligned} 0 &\leq g_x(y) - \frac{1}{2}\|\nabla g_x(y)\|^2 \\ &= f(y) - f(x) + \langle \nabla f(x), x - y, \rangle - \frac{1}{2}\|\nabla f(y) - \nabla f(x)\|^2. \end{aligned}$$

Similarly by symmetry we also obtain that for g_y that

$$0 \leq f(x) - f(y) + \langle \nabla f(y), y - x, \rangle - \frac{1}{2}\|\nabla f(x) - \nabla f(y)\|^2.$$

Now adding the two inequalities and rearranging we obtain, for any $x, y \in \mathcal{H}$,

$$\|\nabla f(y) - \nabla f(x)\|^2 \leq \langle \nabla f(y) - \nabla f(x), y - x, \rangle,$$

showing that ∇f is firmly non-expansive. □

Finally we have this last result that ensures the validity of the proximal algorithm presented in section 4.2.1 below.

Theorem 4.8. *Let I be the identity and f a convex function with L-Lipschitz gradient. The operator $I - \delta\nabla f$ is averaged for any $\delta \in (0, 2/L)$.*

Proof. By Theorem 4.7 $-\nabla f/L$ is firmly non expansive hence by Lemma 4.5 $1/2$-averaged i.e. $-\nabla f/L = (I + N)/2$ for some non expansive operator N. Then since

$$I - \frac{2}{L}\nabla f = I - (I + N)/2 - (I + N)/2 = -N$$

we see that $I - 2\nabla f/L$ is in fact non expansive. For $\alpha \in [0, 1)$ we may write

$$\alpha I + (1 - \alpha)(I - \frac{2}{L}\nabla f) = \alpha I + (1 - \alpha)I - (1 - \alpha)\frac{2}{L}\nabla f = I - (1 - \alpha)\frac{2}{L}\nabla f$$

showing that with $\delta := 2(1 - \alpha)/L \in [0, 2/L)$, $I - \delta\nabla$ averaged □

Next we will introduce the proximity operator and see how to use it to express the general forward-backward operator for g a proper and convex function.

4.2 Proximity operator based algorithms

Next define $p_{\delta g} : \mathcal{H} \to \mathcal{H}$ by

$$p_{\delta g}(y) := \underset{x \in \mathcal{H}}{\arg\min} \Big\{ \frac{1}{2\delta} \|x - y\|_2^2 + g(x) \Big\}, \tag{4.6}$$

for any proper convex function g and $\delta > 0$. The operator p_g is the so called ***proximity operator*** introduced in Moreau (1962) and independently in Minty (1962). We note that

$$x \in p_{\delta g}(y) \Leftrightarrow 0 \in x - y + \delta \partial g(x) \Leftrightarrow y \in (I + \delta \partial g)x$$

with I the identity in $\mathcal{H}$. Now as the g is convex and the 2-norm is strongly convex it follows that the problem in (4.6) has a unique minimum hence $p_{\delta g}$ is singled valued implying

$$y = (I + \delta \partial g)x. \tag{4.7}$$

Furthermore using a result in Minty (1962) for general maximal monotone operators, Moreau (1965) shows that ***all*** $z \in \mathcal{H}$ can be uniquely decomposed as in (4.7) given suitable $x \in \mathcal{H}$. Thus the operator $(I + \delta \partial g) : \mathcal{H} \to \mathcal{H}$ is surjective and single valued. Especially the inverse operator $(I + \delta \partial g)^{-1} : \mathcal{H} \to \mathcal{H}$ is surjective and single-valued implying that

$$p_{\delta g} = (I + \delta \partial g)^{-1}.$$

Thus the proximity operator is in fact the resolvent of the sub-differential operator implying that it is in fact firmly non-expansive as in Definition 4.2. To see this let x, y respectively x', y' satisfy (4.7). Then

$$\begin{aligned} \langle p_{\delta g}(y) - p_{\delta g}(y'), y - y' \rangle &= \langle x - x', (I + \delta \partial g)x - (I + \delta \partial g)x' \rangle \\ &= \|x - x'\|_2^2 + \langle x - x', \delta \partial g(x) - \delta \partial g(x') \rangle \\ &\geq \|p_{\delta g}(y) - p_{\delta g}(y')\|_2^2 \end{aligned}$$

using that ∂g is a monotone operator. We also see that the forward-backward operator (4.4) can be written as the composition

$$(I + \delta \partial g)^{-1}(I - \delta \nabla f) = p_{\delta g}(I + \delta \nabla f).$$

In particular, this shows both the metric projection and the soft thresholding operator are firmly non-expansive as they are special cases of the proximity operator.

Example 4.9. Let $C \subset \mathcal{H}$ denote a non-empty closed convex set and let f be given by the Dirac delta function for C, $g := \delta_C$, which is equal to ∞ for $x \in C$ and zero otherwise. It follows that

$$p_g(x) = pr_C(x) := \underset{C}{\arg\min} \|x - z\|^2,$$

where pr_C denotes the metric projection of x onto C. Hence the proximity operator is a generalization of the metric projection onto a convex set C. Especially the properties discussed above are immediate for the metric projection operator pr_C. ◦

We may also view the proximity operator as a penalized projection operator.

Example 4.10. Consider the convex function $x \mapsto g(x) := \lambda\|x\|_1$, $\lambda > 0$. Evaluating the proximity operator amounts to sub differentiating and setting to zero i.e.

$$0 \in \frac{\partial}{\partial x}\Big(\frac{1}{2}\|x - z\|_2^2 + \lambda\|x\|_1\Big)\Big|_{x=x^*} = x^* + [-\lambda - z, \lambda - z]$$

hence

$$x^* = S(z, \lambda) := \begin{cases} z + \lambda & , z \in (-\infty, -\lambda) \\ 0 & , z \in [-\lambda, \lambda] \\ z - \lambda & , z \in (\lambda, \infty) \end{cases}$$

where the operator S is the so called *soft thresholding operator*. for the ℓ_1-penalty the proximity operator is given in closed form by the soft thresholding operator. ∘

4.2.1 Proximal gradient based algorithms

Given the results in sections 4.1 and 4.2 the following result is immediate.

Theorem 4.11. *Let f be convex with L_f-Lipschitz continuous gradient and g proper and convex. Then for $x \in \mathcal{H}$ the sequence $(x_n)_n$ where*

$$x_n = (p_{\delta g}(I - \delta\nabla f))^n x$$

converges to a fix point for $p_{\delta g}(I - \delta\nabla f)$ hence a solution to the probelm (4.2).

Proof. To see this we note that, by Theorem 4.8, the operator $I - \delta\nabla f$ is averaged as long as $\delta \in (0, 2/L_f)$. Also as shown in section 4.2 the proximity operator is firmly non-expansive hence $1/2$-averaged by Lemma 4.5. Then as the composition of averaged operators is again an averaged operator, by Theorem 4.3, it follows that $p_{\delta g}(I - \delta\nabla f)$ is averaged. Finally by Theorem 4.4 by iteratively applying an averaged operator to any element in its domain we obtain a sequence that converges to a fixed point given that a fix point exists. □

Thus if a solution x^* to (4.2) exists, we are guaranteed to find it by iteratively applying the proximity operator as long as ∇f is Lipschitz. This implies that if the proximity operator is easy to evaluate we have an easy way of solving (4.2). For instance for ℓ_1-penalized problems, solving (4.2) amounts to iteratively soft thresholding, by example 4.10.

From Theorem 4.11 we obtain the following simple algorithm for solving (4.2).

Algorithm 1 Proximal gradient algorithm

Require: $\delta \in (0, 2/L_f)$, $x_0 := x \in \mathcal{H}$.
1: **for** $k = 0$ to $K \in \mathbb{N}$ **do**
2: $x_n := p_{\delta g}(x_{n-1} - \delta\nabla f(x_{n-1}))$.
3: **if** convergence criterion is satisfied **then**
4: break
5: **end if**
6: **end for**

In Beck and Teboulle (2009) it is verified that the convergence rate of the proximal gradient algorithm is $O(1/k)$ i.e. in k iterations we will be within $O(1/k)$ form the optimal objective value. However, building on Nesterov (1983), Beck and Teboulle (2009)

propose a small modification of the proximal gradient algorithm that improve the convergence rate substantially. The authors call this proposed modification for *fast iterative soft thresholding (FISTA)* even though their algorithm applies to the general problem (4.2) with a general proximity operator. The modified algorithm is as follows.

Algorithm 2 Fast proximal gradient algorithm

Require: $\delta \in (0, 2/L_f)$, $x_0 := x \in \mathcal{H}$, $t_1 := 1$, $y_2 := x_1$.
1: **for** $k = 0$ to $K \in \mathbb{N}$ **do**
2: $x_n := p_{\delta g}(y_n - \delta \nabla f(y_n))$,
$t_{n+1} := (1 + \sqrt{1 + 4t_n^2})/2$,
$y_{n+1} := x_n + (t_n - 1)/t_{n+1}(x_n - x_{n-1})$.
3: **if** convergence criterion is satisfied **then**
4: break
5: **end if**
6: **end for**

In Beck and Teboulle (2009) they show that this algorithm has a convergence rate of $O(1/k^2)$. Furthermore as noted by the authors this is the optimal convergence rate for a first order method according to Nemirovsky and Yudin (1983).

We know that to ensure convergence of the proximal algorithms we have to restrict the step size i.e. $\delta \in (0, 2/L_f)$. Especially we need to know the Lipschitz constant L_f of the gradient ∇f. In many cases it is not possible to obtain the Lipschitz constant and in turn determine an appropriate step size. Under such circumstances we can infer a backtracking step.

Algorithm 3 Backtracking Line search

Require: y_k, δ_{k-1}, $s \in (0, 1)$, $\delta := \delta_{k-1}$, $z := p_{\delta g}(y_k - \delta \nabla f(y_k))$
1: **if** $f(z) \leq f(y_k) + \langle \nabla f(y_k), z - y_k \rangle + \|z - y_k\|^2/(2\delta)$ **then**
2: Return $\delta_k = \delta$, $x_{k+1} := z$
3: **else**
4: $\delta := s\delta$ and go to step 1.
5: **end if**

4.2.2 Non-Lipschitz loss gradients

We finish our discussion of solution algorithms related to the problem (4.2) by providing two approaches to solving this problem for a non-Lipschitz continuously differentiable loss or likelihood, using a proximal gradient based algorithm. This for instance relevant for the GLM framework from subsection 3.2.1.2 as well as for the maximin effects framework in subsection 3.2.1.2.

4.2.2.1 Quadratic approximation

One approach is to replace f in (4.2) with the an quadratic approximation to it in some point $x^{(k)}$ given by

$$f_{Q_k}(x) := f(x^{(k)}) + \nabla_x f(x^{(k)})^\top (x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^\top H^{(k)} (x - x^{(k)}),$$

where $H^{(k)}$ is the Hessian of f ($H := \nabla^2 f$) evaluated in $x^{(k)}$. This leads to the minimization problem

$$\tilde{x}^{(k+1)} = \arg\min_{x \in \mathbb{R}^p} -\nabla_x f(x^{(k)})^\top (x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^\top H^{(k)}(x - x^{(k)}) + g(x), \tag{4.8}$$

which corresponds to finding the search direction defined by equation (6) in Tseng and Yun (2009) for the gradient based descent algorithm. Solving (4.8) is equivalent to solving a penalized weighted least squares problem. Thus using this approximation we obtain an outer gradient descent loop and may use the proximal gradient algorithm as an inner loop when solving (4.8). In Lund et al. (2017) we call this the GD-PG algorithm.

Algorithm 4 GD-PG

Require: $x^{(0)}$
1: **for** $k = 0$ to $K \in \mathbb{N}$ **do**
2: given $x^{(k)}$ obtain f_{Q_k}.
3: specify the proximal stepsize δ_k
4: given $x^{(k)}, \delta_k$: solve (4.8) using the a proximal gradient loop.
5: given $x^{(k)}, \tilde{x}^{(k+1)}$: use a line search to compute $x^{(k+1)}$
6: **if** convergence criterion is satisfied **then**
7: break
8: **end if**
9: **end for**

Given regularity conditions this algorithm will produce a sequence of iterates that will converges to a critical point for the objective function in (4.2) as shown in Lund et al. (2017) where the approach is equivalent to iterated penalized least squares.

4.2.2.2 A non-monotone proximal based algorithm

Another approach is to solve the problem directly without approximating the loss. This is the approach taken in Lund et al. (2017) where a softmaximin loss function approximating the loss in subsection 3.2.1.2 is minimized using a so called non-monotone proximal gradient algorithm where the loss only need to have a gradient that is locally Lipschitz continuously differentiable. For completeness we give the non-monotone proximal gradient algorithm from Chen et al. (2016) here.

Algorithm 5 NPG

Require: x^0, $L_{max} \geq L_{min} > 0$, $\tau > 1$, $c > 0$, $M \geq 0$ arbitrarily
1: **for** $k = 0$ to $K \in \mathbb{N}$ **do**
2: choose $L_k \in [L_{min}, L_{max}]$
3: solve $x := p_{g/L_k}(x^{(k)} - \frac{1}{L_k}\nabla f(x^{(k)}))$,
4: **if** $F(x) \leq \max_{[k-M]_+ \geq i \geq k} F(x^{(i)}) - c/2\|x - x^{(k)}\|$ **then**
5: $x^{k+1} := x$
6: **else**
7: $L_k := \tau L_k$ and go to 3.
8: **end if**
9: **end for**

Here step 3 is the proximal step with the proximal operator defined in (4.6). We also note that for $M = 0$ the algorithm is in fact monotone.

We note that this algorithm in particularly applies when ever the loss is strongly convex, see Definition 4.1, as in this case the assumptions from Chen et al. (2016) are all satisfied, as shown in Lund et al. (2017). We note that it should also be possible to solve the estimation problem for the GLM class, i.e. minimize the loss (3.12), using this algorithm.

4.3 Array-tensor computations

We have not thus-far discussed why we have chosen to work with proximal gradient based algorithms. After all there are several different approaches to solving a problem like (4.2). For instance, in the context of non-differentiable penalization one of the most efficient strategies is seemingly to turn the original problem into a sequence of penalized weighted least square problems as discussed in subsection 4.2.2.1 and then solve each of these subproblems using a coordinate descent algorithm. A very efficient and remarkably robust implementation of this approach is the `glmnet` solver from Friedman et al. (2010). This routine solves the problem (4.2) with g a convex combination of the ℓ_1 and the ℓ_2 norms (the so called *elastic net penalty*) for a variety of models in the GLM framework. Apparently this routine derives its efficiency form a happy marriage between the induced sparsity from the ℓ_1-penalty and the marginal nature of the coordinate descent algorithm. Especially it exploits the sparse nature of the problem by using a so called active set strategy to only optimize over non-zero (active) parameter coordinates. Furthermore the number of operations performed at each iteration is quite limited even for large scale data problems as the algorithm only needs to update coordinate-wise, i.e. never needs to perform the entire (design) matrix-vector products. Especially as shown in Hastie et al. (2015) this algorithm can actually outperform proximal gradient based algorithms for models in the GLM framework.

However, the efficiency of the coordinate algorithm depends on having easy access to the columns of the design matrix hence in turn relies on the entire design matrix. For the model framework we consider i.e. models that have a multi component array tensor structure as discussed in section 3.2.2 the (tensor structured) design matrix in large-scale applications will become prohibitively large. Thus in order to solve large-scale MCAT problems we need a design matrix free algorithm which in turn makes the coordinate descent algorithm, seem like a less attractive candidate. Furthermore as we will discuss next, within the MCAT framework you can perform design matrix vector multiplication without constructing the design matrix and furthermore this multiplication is highly efficient compared to an ordinary matrix-vector multiplication. Thus in some sense there is an inherent mismatch between the coordinate descent algorithm and the MCAT framework as the coordinate descent algorithm never evaluates the matrix-vector product but on the other relies heavily on having access to the entire design matrix.

Instead in order to exploit the structure of MCAT models we need an algorithm that can efficiently solve problems like (4.2) and relies on design matrix-vector multiplication when doing so. Now, considering the proximal gradient based algorithms in section 4.2 we note that each iteration involves

1. evaluation of the gradient ∇f.
2. evaluation of the proximity operator $p_{\delta g}$.
3. evaluation of the objective function F.

Given that we can effectively evaluate the proximal operator, e.g. as in example 4.10, the performance of our algorithm will depend on the complexity of evaluating the gradient and the objective. Now for the linear models from section 3.2.1 this essentially comes down to carrying out a design matrix-vector product.

4.3.1 The tensor array computations

We will now briefly introduce the type of computations that form the basis of our algorithm. These computations apparently dates back to Yates (1937). The form considered here is essentially due to Pereyra and Scherer (1973) and De Boor (1979) and later Buis and Dyksen (1996) and Currie et al. (2006).

We start by considering $d = 2$. Let $M = M_2 \otimes M_1$ be a tensor matrix where M_i is $r_i \times c_i$ and A a 2-array (i.e. matrix) of size $c_1 \times c_2$. Then it is a well know property (see Searle (1982)) that

$$M\text{vec}(A) \equiv M_1 A M_2^\top. \tag{4.9}$$

The equivalence symbol $\equiv$ signifies that the left hand side and the right side contain the same values but that these are organized differently. Especially the left hand side is a $r_1 r_2 \times 1$ vector and the right hand side is a $r_1 \times r_2$ matrix, that is whenever M_2 is a matrix with more than two rows or columns.

There are several points to notice about (4.9). First we see that at least for $d = 2$ we may actually compute the matrix vector product $M\text{vec}(A)$ without having access to the matrix M. We also notice that the matrix on the right hand side is computed sequentially via two matrix-matrix products where the first product (from the right) takes $c_1 c_2 r_2$ multiplications and the second takes $r_1 c_1 r_2$ multiplications. On the other hand the direct matrix-vector product takes $r_1 r_2 c_1 c_2$ multiplications and since

$$r_1 c_1 r_2 c_2 > c_1 c_2 r_2 + r_1 c_1 r_2 \Leftrightarrow 1 > \frac{1}{r_1} + \frac{1}{c_2}$$

the sequential matrix-matrix computations, on the right, are actually more efficient than the direct matrix-vector product on the left when $r_1 > 2$ and $c_2 > 2$.

We also note that the computational complexity of the computations on the right apparently depends on how M is organized. If for instance the row dimension r_2, say doubles, then the number of multiplications on the right doubles. However, if the row dimension r_1, doubles then only one of the terms on the right, namely $r_1 c_1 r_2$, doubles. This suggests that for $d = 2$ optimally, the marginal matrices with the largest row dimension should be last in the tensor product.

Next, in order to generalize (4.9) to $d > 2$ dimensions we follow Currie et al. (2006). To do this we need an operator that maps a $c_1 \times \cdots \times c_d$ d-array A to a $c_1 \times c_2 \cdots c_d$ matrix M containing the same values. We denote this operator by mat and note that it simply works by flattening dimensions 2 to d of A. We also need an inverse map denoted mat^{-1} i.e. the map that take a $c_1 \times c_2 \cdots c_d$ matrix M to the $c_1 \times \cdots \times c_d$ d-array A containing the same values.

First we define the d-array equivalent of the sequential matrix-matrix computation in (4.9).

Definition 4.12. *For a matrix M of size $r \times c_1$ and a d-dimensional array A of size $c_1 \times \cdots \times c_d$ the H_M-**transform** of A is the d-dimensional array of size $r \times c_2 \times \cdots \times c_d$ defined by*

$$A \mapsto H_M(A) := \text{mat}^{-1}(M\,\text{mat}(A)).$$

In words, H_M works by flattening the d-array A into a matrix, then computes a matrix-matrix product and then reinstates the array structure given by the first dimension of M and dimension 2 to d of A.

Next we need to generalize the transposition appearing in (4.9) to d-dimensional arrays as follows.

Definition 4.13. *For a d-dimensional array A of size $c_1 \times \cdots \times c_d$ the **rotation** of A denoted $R(A)$ is the d-dimensional array of size $c_2 \times c_3 \times \ldots \times c_d \times c_1$ obtained by permuting the indices of A correspondingly.*

Now by composing the above two definitions we can define how one step in the sequential computations in (4.9) is performed for general d.

Definition 4.14. *For a matrix M of size $r \times c_1$ and a d-dimensional array A of size $c_1 \times \cdots \times c_d$ the **rotated H_M-transform** of A is the d-dimensional array of size $c_2 \times c_3 \times \ldots \times c_d \times r$ defined by*

$$A \mapsto \rho(M, A) = R(H_M(A)).$$

Finally, for any d with A a $c_1 \times \cdots \times c_d$-array and $M = M_d \otimes \cdots \otimes M_1$, M_i is $r_i \times c_i$, we have the following generalization of (4.9)

$$M\text{vec}(A) \equiv \rho(M_d, \rho(M_{d-1}, \rho(\ldots, \rho(M_1, A)\ldots))). \tag{4.10}$$

The mathematical justification of this equivalence can be fund in Currie et al. (2006).

Using (4.10) we can carry out the design matrix-vector multiplications in the MCAT framework without having access to the overall design matrix. This drastically reduces the memory foot print of any algorithm that exploits these type of structure. Especially, the amount of memory needed to store the over all design matrix from subsection 3.2.2 is of the order $\prod_i K_i N_i$ while the amount of memory required to store the marginal components is only of order $\sum_i K_i N_i$. Furthermore as noted in Buis and Dyksen (1996) for $M = M_d \otimes \cdots \otimes M_1$ and $r_i = c_i = m$ for all i (square matrices) the direct matrix-vector product is $O(m^{2d})$ while the corresponding array computation is $O(dm^{d+1})$. Hence we can expect substantial gains in terms of computational efficiency by applying the tensor array computations instead of a direct matrix-vector computations. This shows that in principle we can speed up any algorithm employing the direct matrix-vector product for a tensor matrix.

It is these computations that motivates the MCAT framework and which are exploited throughout in the following manuscripts for several different settings. By combining these computations with a proximal gradient based solution algorithm we obtain solution algorithms for MCAT structured models that scale very well with the size of the data.

Part II

Manuscripts

Chapter 5

Penalized estimation in large-scale generalized linear array models

Lund, A., M. Vincent, and N. R. Hansen (2017). Penalized estimation in large-scale generalized linear array models. *Journal of Computational and Graphical Statistics*, To appear.

Penalized estimation in large-scale generalized linear array models

Adam Lund*
Department of Mathematical Sciences, University of Copenhagen,
Martin Vincent†
Department of Mathematical Sciences, University of Copenhagen
and
Niels Richard Hansen*
Department of Mathematical Sciences, University of Copenhagen

January 3, 2017

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 paper 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.

Keywords: penalized estimation, generalized linear array models, proximal gradient algorithm, multidimensional smoothing

*Part of the Dynamical Systems Interdisciplinary Network, University of Copenhagen.

†Supported by The Danish Cancer Society and The Danish Strategic Research Council/Innovation Fund Denmark.

1 Introduction

The generalized linear array models (GLAMs) were introduced in Currie et al. (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 et al. (2006) – that will also be central to this paper – is multivariate smoothing where data is observed on a multidimensional grid.

In this paper we present results on 3-dimensional smoothing for two quite different real data sets where the aim was to extract a smooth mean signal. The first data set contains voltage sensitive dye recordings of spiking neurons in a live ferret brain and was modeled in a Gaussian GLAM framework. The second data set contains all registered Medallion taxi pick ups 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 non-penalized fit, when trying to predict missing observations. Other potential applications include factorial designs and contingency tables.

Currie et al. (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 in Currie et al. (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 non-quadratic penalty functions like the ℓ_1-penalty or for that matter non-convex penalty functions like the smoothly clipped absolute deviation (SCAD) penalty.

In this paper 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 Sections 2.8.5 and 2.8.6 in Bühlmann and van de Geer (2011) our algorithm can easily be extended to handle non-convex 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 paper are, however, only valid for a convex penalty.

Algorithms considered in the literature hitherto for ℓ_1-penalized estimation in GLMs, see e.g. 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 paper 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 paper 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

$$\eta(\theta) \coloneqq X\theta \tag{1}$$

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

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

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

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

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

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

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

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

where $-l$ is continuously differentiable.

For a GLAM the vector y is assumed given as $y = \text{vec}(Y)$ (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

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

We let $\langle X_{r,j} \rangle \coloneqq \langle X_{1,1}, \ldots, X_{c,d} \rangle$ 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,

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

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

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

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

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 et al. (2006). For non-quadratic 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 squares 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

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

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

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

denote the penalized weighted least squares estimate and define

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

where the stepsize $\alpha_k > 0$ is determined to ensure sufficient descent of the objective function, e.g. 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

$$x^* \coloneqq \arg\min_{x \in \mathbb{R}^p} h(x) + \lambda J(x), \tag{12}$$

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

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

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

$$y \coloneqq x^{(l)} + \omega_l \Big(x^{(l)} - x^{(l-1)} \Big) \quad \text{and} \tag{13}$$

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

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, e.g. $\omega_l = (l-1)/(l+2)$, 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

$$h(\theta) \coloneqq \frac{1}{2n} \| \sqrt{W^{(k)}} (X\theta - z^{(k)}) \|_2^2 \tag{15}$$

we have that

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

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 below. 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 data sets.

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

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

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

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

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 $\Phi \coloneqq (\phi_m((x_1, \ldots, x_d)_i))_{i,m}$, 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

$$\phi_{m_1,\ldots,m_d} \coloneqq \phi_{1,m_1} \otimes \phi_{2,m_2} \otimes \cdots \otimes \phi_{d,m_d}, \tag{18}$$

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 $n \times p$ ($p \coloneqq \prod_{j=1}^{d} p_j$) tensor product matrix

$$\Phi = \Phi_d \otimes \cdots \otimes \Phi_1 \tag{19}$$

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 data sets 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 data sets 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,

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

with $\hat{\theta}^{\texttt{x}}$ 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 data set 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 data set 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 3-dimensional array data set is $25 \times 25 \times 977$ corresponding to $n = 610,625$ data points.

As basis functions we used cubic B-splines with $p_j \coloneqq \max\{[n_j/5], 5\}$ basis functions in each dimension (see Currie et al. (2006) or Wood (2006)). This corresponds to a parameter array of size $5 \times 5 \times 196$ ($p = 4,900$) and a design matrix of size $610,625 \times 4,900$ for the entire data set. 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 data set as well as to subsets of the data set 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 data set 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 data set 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 data set consisting of triples containing longitude, latitude and date-time of the pickup. First we cropped the

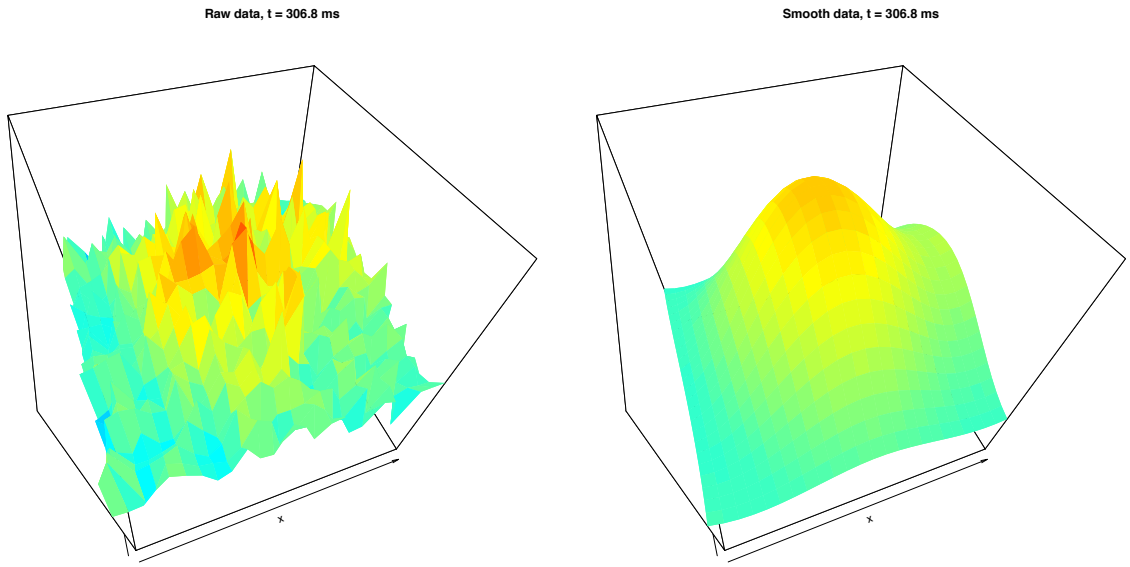


Figure 1: The raw neuron data (left) and the smoothed fit (right) after 306.8 ms. The supplementary material contains movies of the complete raw data and smoothed fit.

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 was rotated and summarized as binned counts in $100 \times 100 \times 168$ spatial-temporal bins. Each temporal bin represents one hour. The data was then further cropped to cover Manhattan only, which removed the large black parts – as seen on Figure 3 – where pickups were rare. The total size of the data set 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 $p = 7,938$ and a design matrix of size $449,064 \times 7,938$ for the entire data set. 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 data set as well as to subsets of the data set that correspond to smaller design matrices.

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 data set using `glmnet`, and results for `glmnet` are only reported for models that could be fitted. Except for the smallest design matrix the run

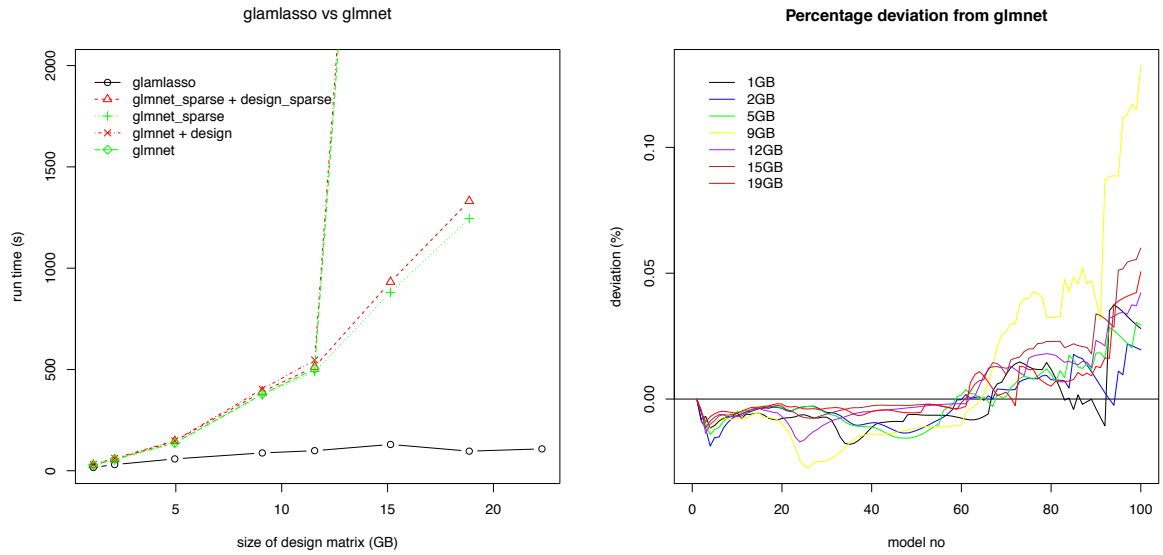


Figure 2: Benchmark results for the neuron data. Run time in seconds is shown as a function of the size of the design matrix, when not stored in sparse format, in GB (left). Relative deviation in the attained objective function values as given by (20) is shown as a function of model number (right), where a larger model number corresponds to less penalization (smaller λ).

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 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

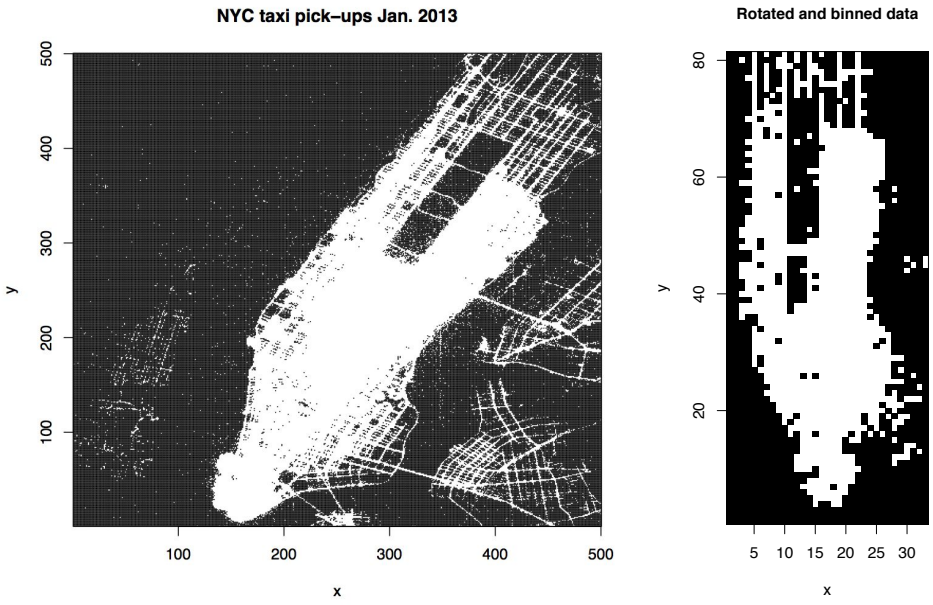


Figure 3: Binned counts of registered NYC taxi pickups for January 2013 using 500×500 spatial bins (left) and the same data rotated, binned to 100×100 spatial bins and cropped to cover Manhattan only (right).

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 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.

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.

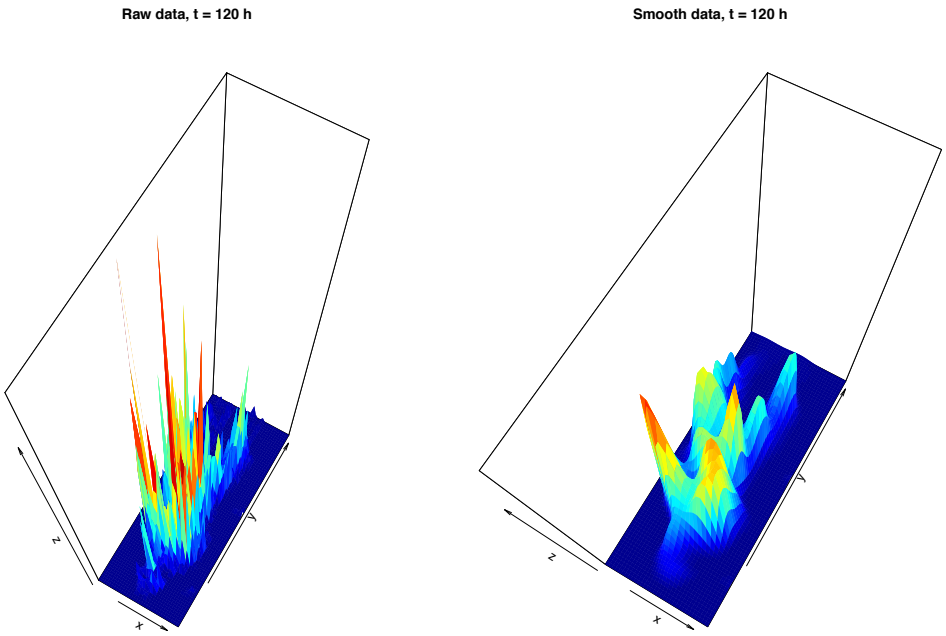


Figure 4: The raw NYC taxi data (left) and the smoothed fit (right) around midnight on Saturday, January 5, 2013. The supplementary material contains movies of the complete raw data and smoothed fit.

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

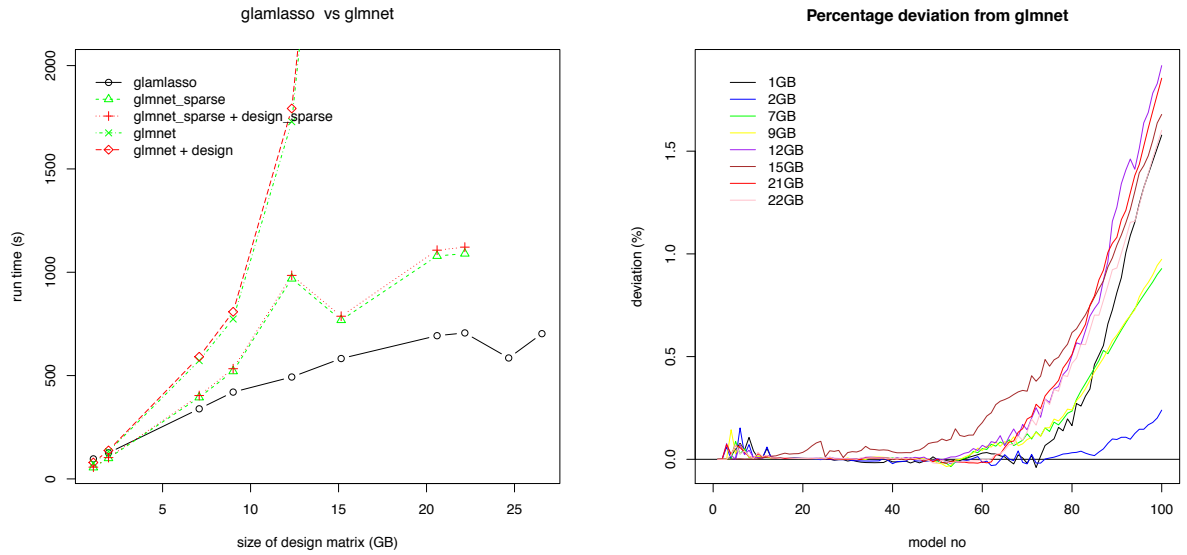


Figure 5: Benchmark results for the taxi data. Run time in seconds is shown as a function of the size of the design matrix, when not stored in sparse format, in GB (left). Relative deviations in the attained objective function values as given by (20) is shown as a function of model number (right), where a larger model number corresponds to less penalization (smaller λ).

$\|\cdot\|_2$ 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 acceleration 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 acceleration sequence considered here corresponds 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 e.g. Theorem 1.1 in Beck and Teboulle (2010). In this case $(x^{(l)})$ always converges towards 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 towards 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).

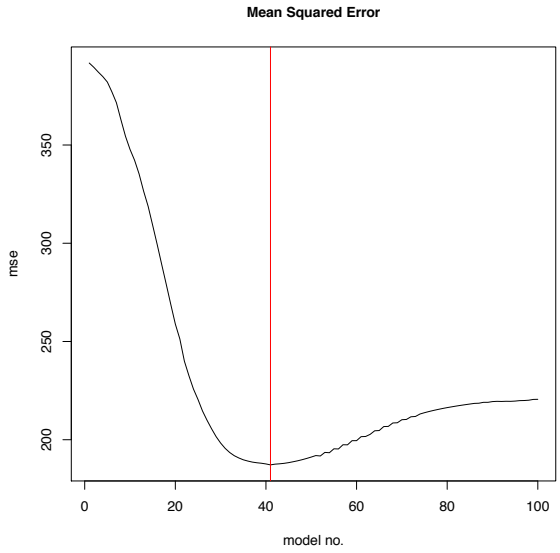


Figure 6: The mean squared error for prediction on grid points left out of the model fitting as a function of model number. The vertical red line indicates the model with minimal MSE (model 41).

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

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

where ϱ denotes the spectral radius.

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

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

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

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

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

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

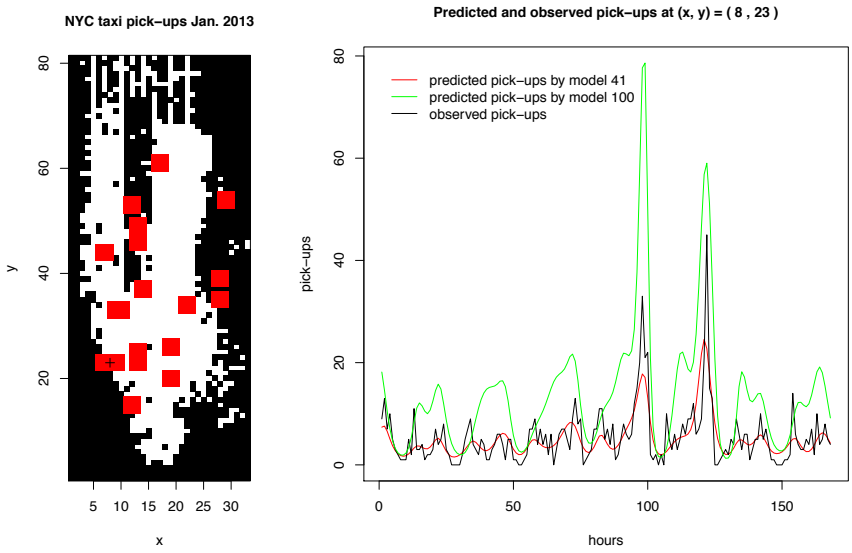


Figure 7: Binned number of NYC taxi pickups as in Figure 3 (left) with red 3×3 squares indicating bins that were removed from the data before model fitting. Predicted and observed number of pickups at spatial bin $(8, 23)$ (indicated with a "+" on the left figure) are shown as a function of time in hours (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.

By the properties of the tensor product we find that

$$X_r^\top X_r = X_{r,1}^\top X_{r,1} \otimes \ldots \otimes X_{r,d}^\top X_{r,d}, \tag{29}$$

whose eigenvalues are of the form $\alpha_{1,k_1}\alpha_{2,k_2}\cdots\alpha_{d,k_d}$, with α_{j,k_j} being the k_jth eigenvalue of $X_{r,j}^\top X_{r,j}$, see e.g. 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 $W^{(k)} = W_d^{(k)} \otimes \ldots \otimes W_1^{(k)}$ then

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

and by similar arguments as in the proof above,

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

The outer loop is similar to the outer loop used in e.g. the R packages `glmnet`, Friedman et al. (2010), and `sglOptim`, Vincent et al. (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 $\alpha_k \coloneqq b^j \alpha_0$, 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

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

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

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

where $v \in (0, 1)$ 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 $d \coloneqq \theta - \theta^{(k)}$ denote a (potential) search direction we see that

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

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 $\Theta_0 \coloneqq \{\theta \mid F(\theta) \leq F(\theta^{(0)})\}$ 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 2 and 3-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

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

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

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

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).

Below 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, e.g. by (34). The arrays $U^{(k)}$ and $Z^{(k)}$ are computed by entrywise computations using (52) and (9), respectively. If the weights given by (34) are used, $Z^{(k)}$ can be computed directly by (54) 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

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

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

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

The matrix products $X_{r,k}^\top W_j^{(k)} X_{m,j}$ for $r, m = 1, \ldots, c$ and $j = 1, \ldots, d$ 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

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

with

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

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

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

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,

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

see e.g. 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,

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

see Parikh and Boyd (2014). For more examples see Parikh and Boyd (2014) and see also Zhang et al. (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 2 and 3-dimensional GLAMs. We note that as the SCAD penalty is non-convex the resulting optimization problem becomes non-convex 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 non-convex 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 utilizes 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 $\lambda_{max} > \ldots > \lambda_{min}$. Here λ_{max} is the infimum over the set of penalty parameters yielding a zero solution to (5) and λ_{min} is a (user specified) fraction of λ_{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 et al. (2006) – as a way to model scattered (non-grid) 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 3-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 a $\mathcal{N}_{p_j}(0, \Sigma)$ 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 non-zero 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

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

where $B_1, \ldots, B_p$ are i.i.d. 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, 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 and Figures 8, 9 and 10 below 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 Guassian 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 minutes and up to one hour) underlining the advantage of our design matrix free method.

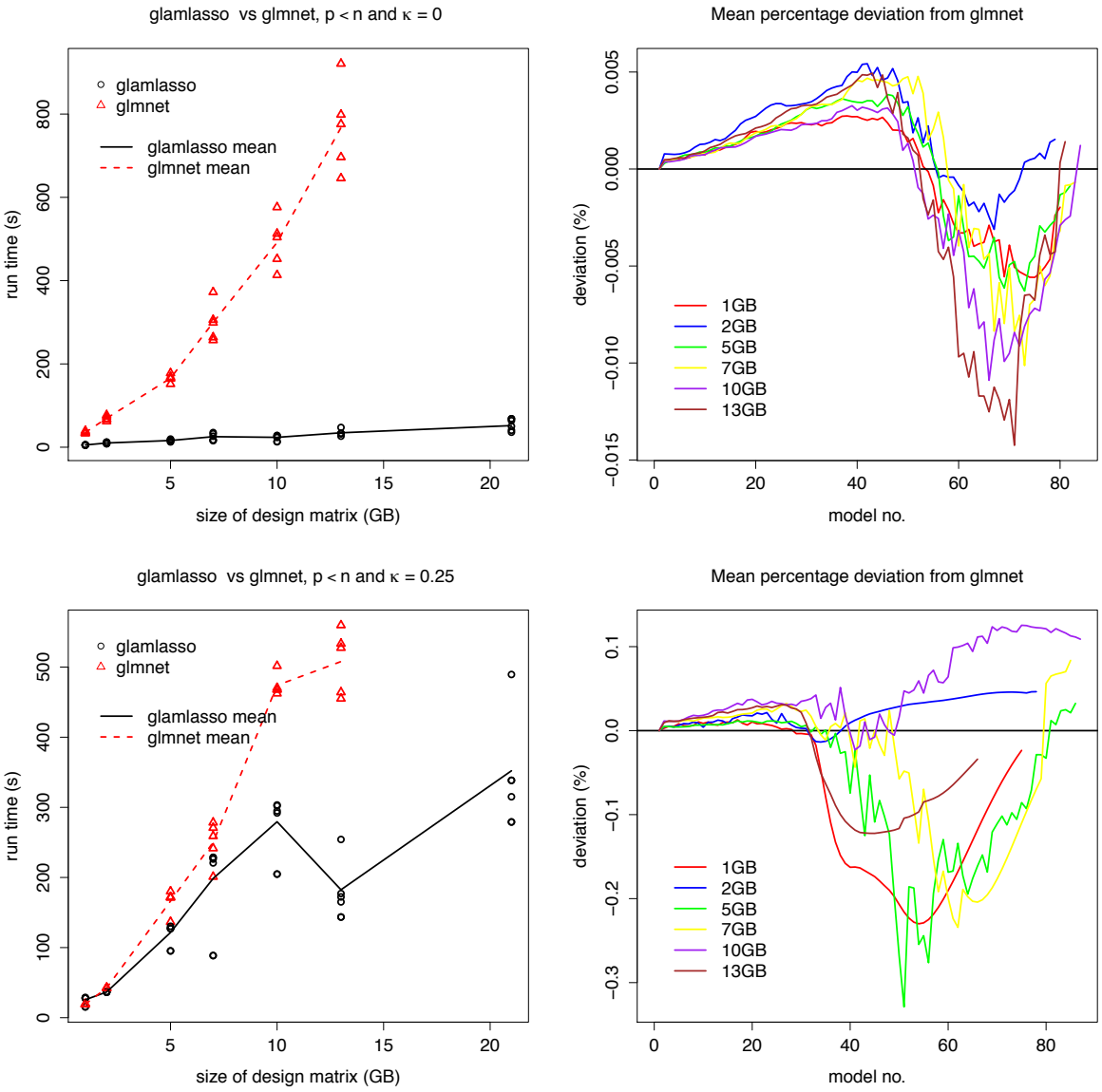


Figure 8: Benchmark results for the Gaussian models and $p < n$. Run time in seconds is shown as a function of the size of the design matrix in GB (left). Relative mean deviation in the attained objective function values as given by (20) is shown as a function of model number (right). The top row gives the results for $\kappa = 0$ and the bottom for $\kappa = 0.25$.

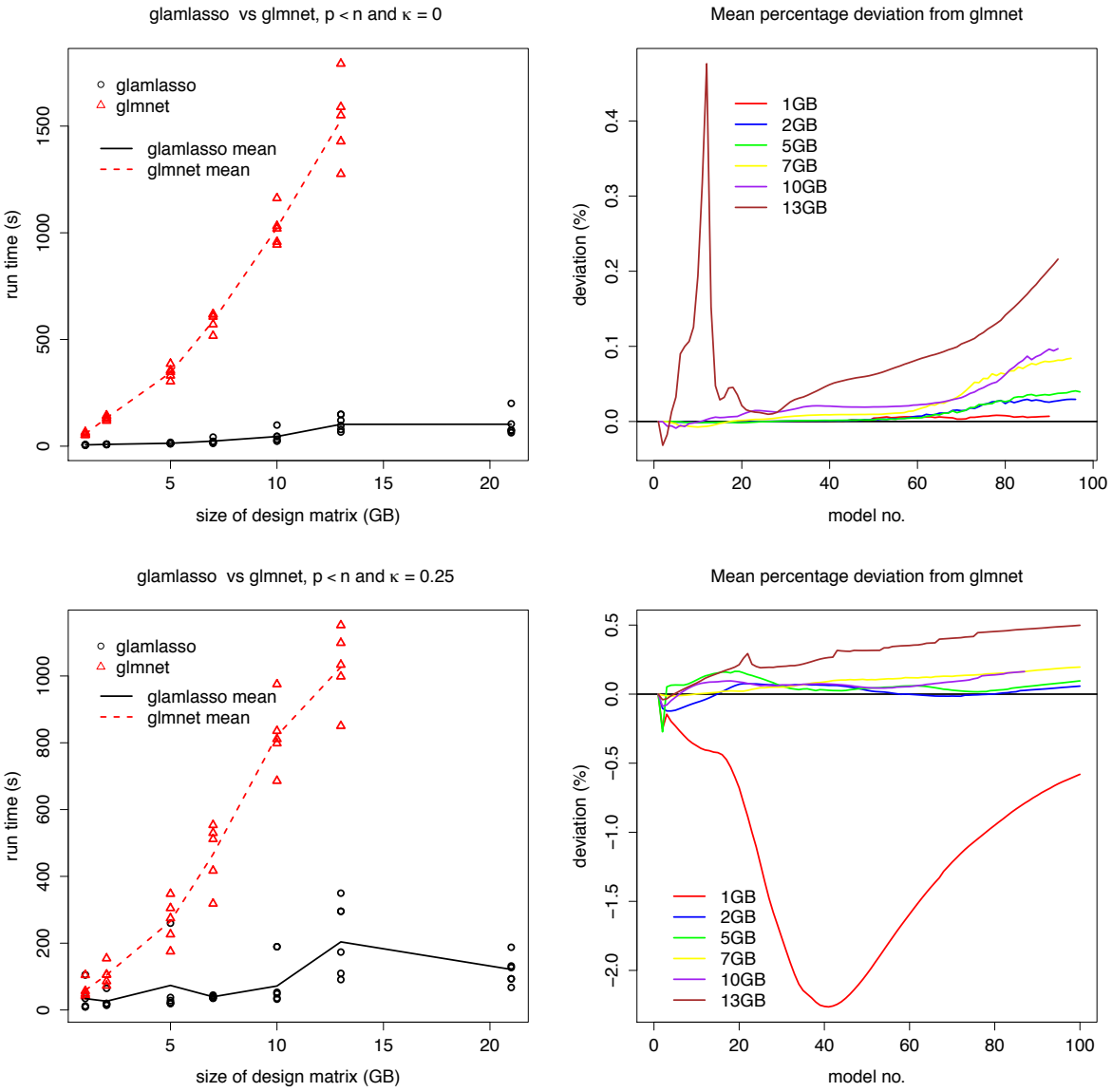


Figure 9: Benchmark results for the Poisson models and $p < n$. Run time in seconds is shown as a function of the size of the design matrix in GB (left). Relative mean deviation in the attained objective function values as given by (20) is shown as a function of model number (right). The top row gives the results for $\kappa = 0$ and the bottom for $\kappa = 0.25$.

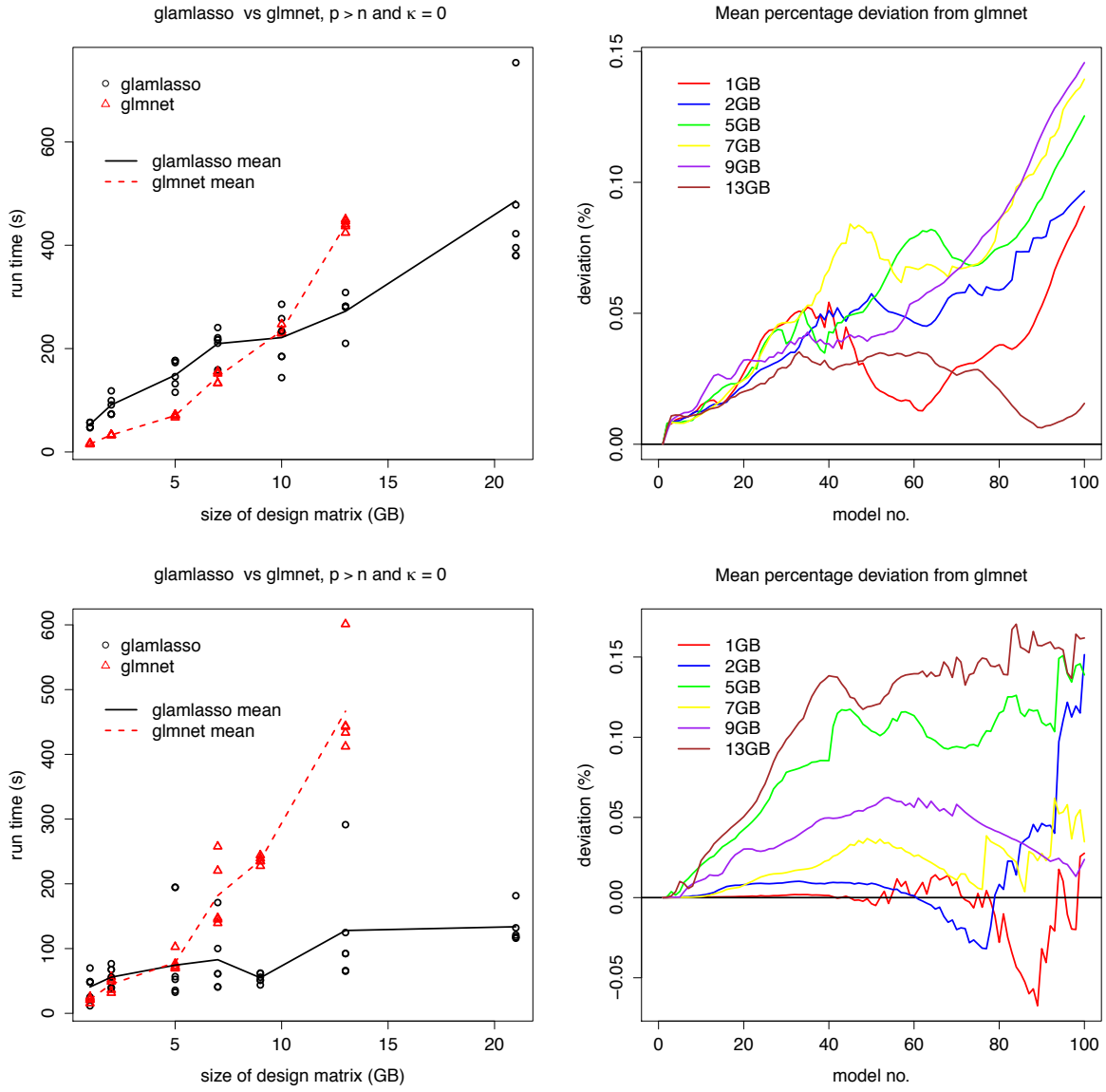


Figure 10: Benchmark results for $p > n$. Run time in seconds is shown as a function of the size of the design matrix in GB (left). Relative mean deviation in the attained objective function values as given by (20) is shown as a function of model number (right). The top row gives the results for the Gaussian model and the bottom for Poisson model.

7 Discussion

The algorithm implemented in the R package `glmnet` and described in Friedman et al. (2010) computes the penalized and weighted least squares 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 straight forward to implement a memory efficient version of the coordinate descent algorithm 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 e.g. 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 acceleration 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 Section 5.5 in Hastie et al. (2015). For large-scale problems and/or structured design matrices – such as the tensor product design matrices considered in this paper – 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 squares 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 of 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 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 paper, the generality allows for the incorporation of convex parameter contraints. 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 dimished 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 non-local 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 et al. (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`.

8 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).

References

Bapat, R. (2010). *Graphs and Matrices*. Universitext. Springer.

Beck, A. and M. Teboulle (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences 2*(1), 183–202.

Beck, A. and M. Teboulle (2010). Gradient-based algorithms with applications to signal recovery problems. In D. P. Palomar and Y. C. Eldar (Eds.), *Convex Optimization in Signal Processing and Communications*, pp. 3–51. Cambridge University Press.

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

Buis, P. E. and W. R. Dyksen (1996). Efficient vector and parallel manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS) 22*(1), 18–23.

Currie, I. D., M. Durban, and P. H. Eilers (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68*(2), 259–280.

De Boor, C. (1979). Efficient computer manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS) 5*(2), 173–182.

Eddelbuettel, D. and R. François (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software 40*(8), 1–18.

Friedman, J., T. Hastie, and R. Tibshirani (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software 33*(1), 1.

Hastie, T., R. Tibshirani, and M. Wainwright (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. CRC Press.

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

Lund, A. (2016). *glamlasso: Penalization in Large Scale Generalized Linear Array Models*. R package version 2.0.1.

Moreau, J.-J. (1962). Fonctions convexes duales et points proximaux dans un espace hilbertian. *C. R. Acad. Sci., Paris 255*, 2897–2899.

Nelder, J. A. and R. W. M. Wedderburn (1972). Generalized linear models. *Journal of the Royal Statistical Society: Series A (General) 135*(3), 370–384.

Parikh, N. and S. Boyd (2014). Proximal algorithms. *Foundations and Trends® in Optimization 1*(3), 127–239.

Roland, P. E., A. Hanazawa, C. Undeman, D. Eriksson, T. Tompa, H. Nakamura, S. Valentiniene, and B. Ahmed (2006). Cortical feedback depolarization waves: A mechanism of top-down influence on early visual areas. *Proceedings of the National Academy of Sciences 103*(33), 12586–12591.

Springer, M. D. and W. E. Thompson (1970). The distribution of products of beta, gamma and gaussian random variables. *SIAM Journal on Applied Mathematics 18*(4), pp. 721–737.

Tseng, P. and S. Yun (2009). A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming 117*(1-2), 387–423.

Vincent, M., Hansen, and N. R. (2014). Sparse group lasso and high dimensional multinomial classification. *Computational Statistics & Data Analysis 71*, 771–786.

Wood, S. (2006). *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis.

Zhang, H., J. Jiang, and Z.-Q. Luo (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*(2), 163–186.

Zou, H. and R. Li (2008). One-step sparse estimates in nonconcave penalized likelihood models. *Annals of statistics 36*(4), 1509.

A The maps vec and ρ

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

$$[i_1, \ldots, i_d] \coloneqq i_1 + n_1((i_2 - 1) + n_2((i_3 - 1) + \ldots n_{d-1}(i_d - 1) \ldots)). \tag{48}$$

Then vec is defined as

$$\operatorname{vec}(A)_{[i_1,\ldots,i_d]} \coloneqq A_{i_1,\ldots,i_d} \tag{49}$$

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

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

$$\rho(X, A)_{i_1,\ldots,i_d} \coloneqq \sum_j X_{i_d,j} A_{j,i_1,\ldots,i_{d-1}}. \tag{50}$$

From this definition it follows directly that

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

where $[i_1, \ldots, i_d]$ denotes the index defined by (48).

B Exponential families

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

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

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

$$\partial_\eta \log f_{\vartheta(\eta),\psi}(y) = a\vartheta'(\eta)(y - g^{-1}(\eta)), \tag{51}$$

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

$$u_i = a_i\vartheta'(\eta_i)(y_i - g^{-1}(\eta_i)), \quad i = 1, \dots n. \tag{52}$$

The weights commonly used when fitting a GLM are

$$w_i = \vartheta'(\eta_i)(g^{-1})'(\eta_i), \tag{53}$$

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 (53), the working response z, see (9), given the linear predictor η can be computed as

$$z_i = a_i(y_i - g^{-1}(\eta_i))g'(g^{-1}(\eta_i)) + \eta_i, \tag{54}$$

which renders it unnecessary to compute the intermediate score statistic.

Chapter 6

Sparse Network Estimation for Dynamical Spatio-temporal Array Models

Lund, A. and N. R. Hansen (2017). Sparse network estimation for dynamical spatio-temporal array models. In preparation.

SPARSE NETWORK ESTIMATION FOR DYNAMICAL SPATIO-TEMPORAL ARRAY MODELS

By Adam Lund[??] and Niels Richard Hansen[*]

University of Copenhagen[†‡]

Neural field models represent neuronal communication on a population level via synaptic weight functions. Using voltage sensitive dye (VSD) imaging it is possible to obtain measurements of neural fields with a relatively high spatial and temporal resolution.

The synaptic weight functions represent functional connectivity in the brain and give rise to a spatio-temporal dependence structure. Methodology for the estimation of such a spatio-temporal dependence structure from VSD imaging data is developed. The dependence structure is expressed via a stochastic functional differential equation, which leads to a vector autoregressive model of the data via basis expansions of the synaptic weight functions and time and space discretization. By using a 1-norm penalty in combination with localized basis functions it is possible to learn a sparse network representation of the functional connectivity of the brain. It is, however, not possible to minimize the objective function via methods that require the explicit construction of a design matrix as this becomes prohibitively large.

We demonstrate that by using tensor product basis expansions, the computation of the penalized estimator via a proximal gradient algorithm becomes feasible. It is crucial for the computations that the data is organized in an array as is the case for the three dimensional VSD imaging data. This allows for the use of array arithmetic that is both memory and time efficient, and which ultimately makes it possible to estimate a sparse network structure for the brain from the VSD imaging data.

1. Introduction. Neural field models are mesoscopic models of the aggregated voltage or activity of a large and spatially distributed population of neurons. The neuronal network is determined by spatio-temporal weight functions in the neural field model, and we will refer to these weight functions as the *propagation network*. This network determines how signals are propagated in the field model. It is of great interest to learn the propagation network from experimental data, which is the inverse problem for neural

[*]First supporter of the project

MSC 2010 subject classifications: Primary , ; secondary

Keywords and phrases: stochastic functional differential equation, sparse estimation, non-differentiable regularization, array model

field models.

The literature on neural fields is vast and we will not attempt a review, but see [3, 9] and the references therein. The typical neural field model considered is a deterministic integrodifferential equation. The inverse problem for the deterministic Amari equation was treated in [2] and [25], and stochastic neural field models was, for instance, treated in Chapter 9 in [9] and in [16]. One main contribution of the latter paper, [16], was to treat a stochastic version of the Amari equation in the well developed theoretical framework of functional stochastic evolution equations.

Despite the substantial literature on neural fields, relatively few papers have dealt directly with the estimation of neural field components from experimental data. Pinotsis et al. [24] demonstrated how a neural field model can used as the generative model within the dynamic causal modeling framework, where model parameters can be estimated from electrophysiological data. The modeling of voltage sensitive dye (VSD) imaging data in terms of neural fields was discussed in [8], and Markounikau et al. [22] estimated parameters in a neural field model directly from VSD imaging data.

In this paper VSD imaging data is considered as well. This imaging technique has a relatively high resolution in time and space and represent *in vivo* propagation of "brain signals" embedded in neuronal activity on a mesoscopic scale, see [26]. A prevalent notion in the neuroscience literature is that the network in the brain connecting the neurons, which in turn is responsible for the propagation of signals, is sparse, and furthermore that the propagation exhibits a time delay. For instance assuming that a spiking neuron is only triggered by neurons lying in a very localized region would imply spatial sparsity. Furthermore, one can also imagine that firing neurons in one region will trigger neurons in different regions only after some amount of time, yielding temporal sparsity and long range dependence, see e.g. [5], [29], [28], [4], [31]. Finally the possibility of feedback waves in the brain, e.g. as suggested in [26], could also be explained by spatio-temporal dynamics depending on more than just the instantaneous past. These considerations lead us to suggest a class of stochastic neural field models that allows for time delay, and a proposed estimation methodology that provides sparse nonparametric estimation of synaptic weight functions. Thus we do not make assumptions about spatial homogeneity or isotropy of the functional connectivity, nor do we assume that the signal propagation is instantaneous.

In order to derive a statistical model that, from a computational viewpoint, is feasible for realistically sized data sets, a time and space discretized version of the infinite dimensional dynamical model is obtained by replacing

the various integrals with Riemann-Ito type summations and relying on an Euler scheme approximation. This approximation scheme makes it possible to derive a statistical model with an associated likelihood function amenable to non-differentiable regularization. Especially, we show that by expanding each component function in a tensor product basis we can formulate the statistical model as a type of multi-component linear array model, see [20].

The paper is organized as follows. First we give a more technical introduction to the stochastic dynamic models that form the basis for the paper. Then we present the aggregated results from the application of our proposed estimation methodology to a VSD imaging data set. The remaining part of the paper presents the derivation of the linear array model and the key computational techniques required for the actual estimation of the model using array data. The appendix contains further technical proofs, implementational details and the results from fitting the model to individual trials.

2. A stochastic functional differential equation. The data that we will ultimately consider is given as follows. With $\tau, T > 0$, $\mathcal{T} := [-\tau, T]$ and $N_x, N_y, M, L \in \mathbb{N}$ we record, to each of $N_t := M + L + 1$ time points

$$-\tau = t_{-L} < \ldots < t_0 < \ldots < t_M = T, \tag{2.1}$$

a 2-dimensional rectangular $N_x \times N_y$ image of neuronal activity in an area of the brain represented by the Cartesian product $\mathcal{S} := \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^2$. These images consist of $D := N_x N_y$ pixels each represented by a coordinate (x_i, y_j) lying on a grid $\mathbb{G}^2 \subseteq \mathcal{S}$. To each time point each pixel has a color represented by a value $v(x_i, y_j, t_k) \in \mathbb{R}$. Thus the observations are naturally organized in a 3-dimensional array $v := (v(x_i, y_j, t_k))_{i,j,k}$ where the first two dimensions correspond to the spatial dimensions and the third dimension corresponds to the temporal dimension.

As such it is natural to view v as a discretely observed sample in time and space of an underlying spatio-temporal random field V. Following Definition 1.1.1 in [1] any measurable map $V : \Omega \to \mathbb{R}^{\mathcal{R}}$, with $\mathcal{R} \subseteq \mathbb{R}^d, d \in \mathbb{N}$ a parameter set, is called a $(d, 1)$-random field or simply a d-dimensional random field. Especially, for the brain image data, V is real valued with a 3-dimensional parameter set $\mathcal{R} := \mathcal{S} \times \mathcal{T}$ where $\mathcal{S}$ refers to space while $\mathcal{T}$ refers to time. We emphasize the conceptual asymmetry between these dimensions by calling V a spatio-temporal random field and note that in the following $\mathcal{S}$ can have any dimension $d' \in \mathbb{N}$, corresponding to recording d'-dimensional images. For fixed t, as $V(t) := V(\cdot, \cdot, t) : \mathbb{R}^2 \to \mathbb{R}$ this model will inevitably be a stochastic dynamical model on a function space, that is an infinite dimensional stochastic dynamical model.

Based on the discussion in the introduction above we propose to model the random neural field V via a stochastic functional differential equation (SFDE) with the propagation network incorporated into the drift as a spatio-temporal linear filter (a convolution) with an impulse-response function (convolution kernel) quantifying the network. The solution of the SFDE in a Hilbert space $\mathcal{H}$ is then the underlying time and space continuous model V for the data v.

To introduce the general model more formally let $(\Omega, \mathcal{F}, P)$ be a probability space endowed with an increasing and right continuous family $(\mathcal{F}_t)$ of complete sub-σ-algebras of $\mathcal{F}$. Let $\mathcal{H}$ a reproducing kernel Hilbert space (RKHS) of continous functions over the compact set $\mathcal{S}\times\mathcal{S}$. Suppose $(V(t))_t$ is a continuous $\mathcal{F}_t$-adapted, $\mathcal{H}$-valued stochastic process and let $\mathcal{C} := \mathcal{C}([-\tau, 0], \mathcal{H})$ denote the Banach space of continuous maps from $[-\tau, 0]$ to $\mathcal{H}$. Then $(V_t)_t$ where

$$V_t := (V(t+s))_{s\in(-\tau,0)}, \quad t \geq 0, \tag{2.2}$$

defines a $\mathcal{C}$-valued stochastic process over $\mathbb{R}_+$. We call V_t the τ-memory of the random field $(V(r))_{r\in\mathcal{S}\times\mathcal{T}}$ at time $t \geq 0$.

Let $\mu : \mathcal{C} \times [0, T] \to \mathcal{H}$ be a bounded linear operator and consider the stochastic functional differential equation (SFDE) on $\mathcal{H}$ given by

$$\mathrm{d}V(t) = \mu(V_t, t)\mathrm{d}t + \mathrm{d}W(t). \tag{2.3}$$

Here W is a spatially homogenous Wiener process with spectral measure σ (σ is a finite symmetric measure on $\mathbb{R}^2$) as in [23]. That is, W is a centered Gaussian random field such that $(W(x, y, t))_t$ is a $(\mathcal{F}_t)$-Wiener process for every $(x, y) \in \mathbb{R}^2$, and for $t, s \geq 0$ and $(x, y), (x', y') \in \mathbb{R}^2$

$$E(W(t, x, y)W(s, x,' y')) = (t \wedge s)c(x - x', y - y'). \tag{2.4}$$

Here $c : \mathbb{R}^2 \to \mathbb{R}$, the covariance function defined as the Fourier transform of the spectral measure σ, quantifies the spatial correlation in W.

We note that compared to a typical SDE, the infinitesimal dynamic at time t as described by (2.3) depends on the past via the τ-memory of V in the drift operator μ. Hence processes satisfying (2.3) will typically be non-Markovian. We also note that all the memory in the system is modelled by the drift μ.

The non-Markovian property makes theoretical results regarding existence, uniqueness and stability of solutions to (2.3) much less accessible. Corresponding to section 0.2 in [12], in order to obtain theoretical results, it should be be possible to lift the equation (2.3) form a Markovian SDE

on the Banach space $\mathcal{C}$. Consequently an unbounded linear operator (the differential operator) then appears in the drift. It is outside the scope of this paper to pursue a discussion of the theoretical properties of (2.3). General theoretical results on SDEs on Banach spaces are not abundant, see e.g. [10].

2.1. *Drift operator.* The idea is that the drift operator μ will capture both external input to the system (the brain in our context) as well as the subsequent propagation of this input over time and space. By decomposing the drift operator we obtain drift components responsible for modelling instantaneous effects and propagation effects respectively. To this end we will specify the drift operator by the decomposition

$$\mu(V_t, t) := \mathrm{S}(t) + \mathrm{F}(V_t) + \mathrm{H}(V(t)). \tag{2.5}$$

Here $\mathrm{S} : \mathcal{T} \to \mathcal{H}$, $\mathrm{S}(t)(x, y) := s(x, y, t)$ with $s \in L^2(\mathbb{R}^3, \mathbb{R})$ a smooth function of time and space, models a deterministic time dependent exogenous input to the system. $\mathrm{H} : \mathcal{H} \to \mathcal{H}$, $\mathrm{H}(V(t))(x, y) := h(x, y)V(x, y, t)$ where $h \in \mathcal{H}$, a smooth function of space, captures the short range (instantaneous) memory in the system.

The long range memory responsible for propagating the input to the system over time and space is modelled by the operator $\mathrm{F} : \mathcal{C}([0, \tau], \mathcal{H}) \to \mathcal{H}$ given as the integral operator

$$\mathrm{F}(V_t)(x, y) = \int_{\mathcal{S}} \int_{-\tau}^{0} w(x, y, x', y', r)V(x', y', t + r)\mathrm{d}r\mathrm{d}x'\mathrm{d}y'. \tag{2.6}$$

Here $w \in L^2(E^5, \mathbb{R})$ is a smooth weight function quantifying the impact of previous states on the current change in the random field V. Especially, $w(x, y, x', y', r)$ is the weight by which the change in the field at location (x, y) is impacted by the level of the field at location (x', y') with delay r. With this specification, a solution to (2.3), if it exists, can be written in integral form as

$$\begin{aligned} V(t, x, y) = \int_0^t \Bigg(& s(x, y, u) + \int_{\mathcal{S}} \int_{-\tau}^{0} w(x, y, x', y', r)V(x', y', u + r)\mathrm{d}r\mathrm{d}x'\mathrm{d}y' \\ & + h(x, y)V(u) \Bigg) du + \int_0^t \mathrm{d}W(u, x, y)du. \end{aligned} \tag{2.7}$$

Thus (2.7) characterizes a solution to a stochastic delay differential equation (SDDE) on $\mathcal{H}$ with delays distributed over time and space (spatio-temporal distributed delays) according to the impulse-response function w. We can think of w as quantifying a spatio-temporal network in the brain that governs

how the propagation of the input is to be distributed over time and space, and we will refer to w as the network function.

Next we present an example where a statistical model based on the spatio-temporal SDDE model proposed above is fitted to real high dimensional brain image data. The derivation of this statistical model relies on a space time discretization and is discussed in section 4. We note that key elements in our approach involves expanding the network function (along with the other component functions) using basis functions with compact support in time and space domain. We then apply regularization techniques to obtain sparse estimate (space-time localized) of the network.

3. Brain imaging data. The data considered in this section is part of a larger data set, containing *in vivo* recordings of the visual cortex of ferret brains provided by Professor Per Ebbe Roland. In the experiment producing this data, a voltage sensitive dye (VSD) technique was used to create a sequence of images of the visual cortex, while presenting a visual stimulus, a stationary white square displayed for 250 ms on a grey screen, to the ferret. Thus each recording or trial is a film showing the brain activity in the live ferret brain, see [26]. The experiment was repeated several times for 11 different ferrets producing a large-scale spatio-temporal data set containing around 300 trials (films) each trial containing around 2048 images and each image containing 464 pixels.

The purpose of this experiment was to study the response in brain activity to the stimulus and its propagation of over time and space. This is possible as the VSD technique can produce in vivo recordings on a mesocopic scale with high resolution in space and time.

For the analysis in this section we consider all trials (12 in total) for one animal (0308) for a rectangular subset of pixels, i.e. 12 films consisting of 977 16×16 frames (images). We let $L := 50$ thus allowing delay effects from 51 frames. Time is measured in frames where 1 frame $= 0.6136$ ms thus we model a delay of roughly 31 ms. For each single trial (film) the model is fitted using a lasso regularized linear array model derived in section 4 below. Each of the 12 single trial fits are summarized in the supplemental material.

3.1. *The aggregated stimulus and network estimates.* By mean aggregating the single trial fits over the all trials we obtain one fit based on 12 trials. Here we will focus on the aggregated estimates of the network function and the stimulus function.

We first plot the estimated stimulus surface to time point 426, the estimated time of maximum depolarization across all trials, see Appendix C.

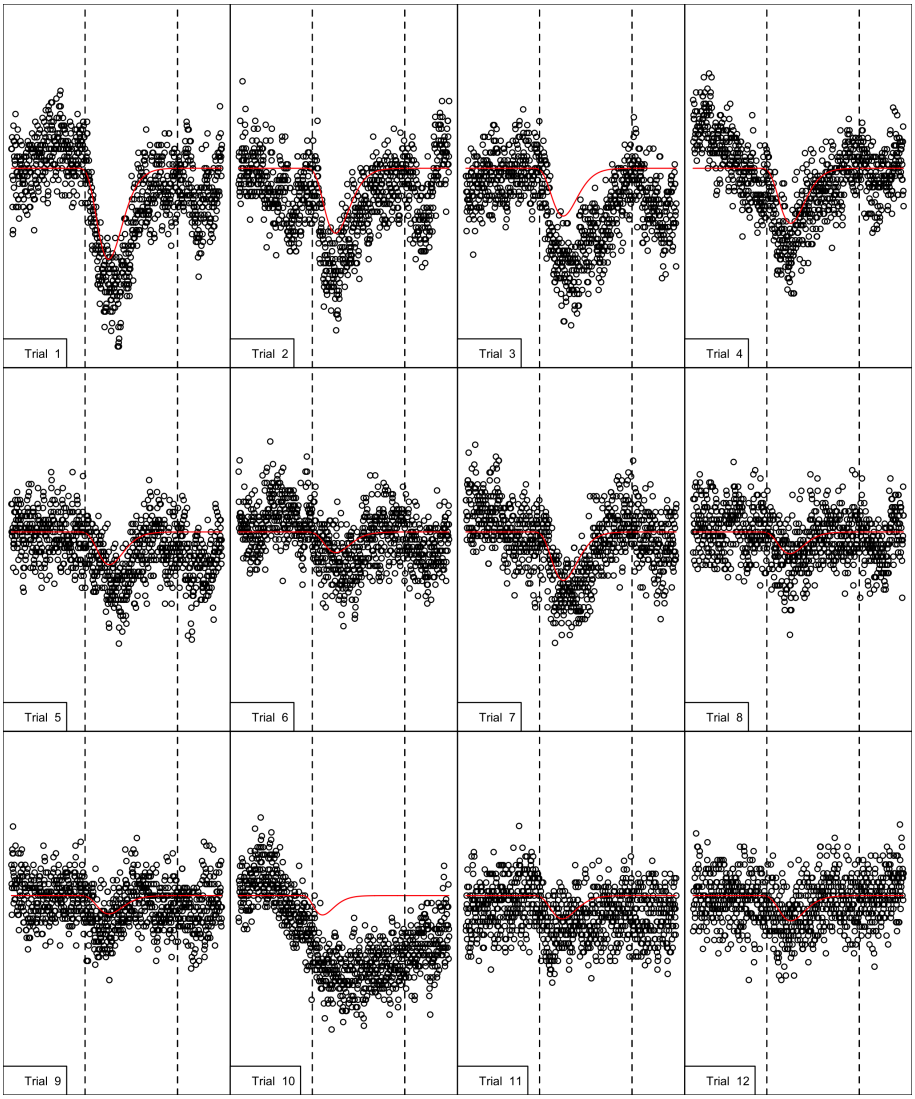


FIG 1. *The data observed at pixel (9,9) for animal 0308 (black) and the estimate of* $s(9, 9, t)$ *(red). Dotted vertical lines indicate stimulus start and stop. Notice the considerable variation among the trials.*

We hypothesize that the "high stimulus" areas visible in Figure 2 correspond to the expected mapping of the center of field of view (CFOV), see Figure 1 in [18].

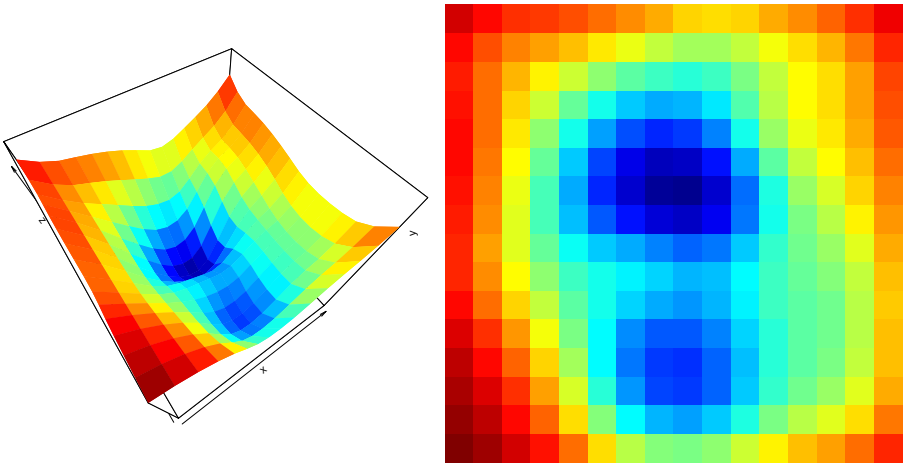


FIG 2. *Mean aggregated fitted stimulus $\hat{s}$ for all pixels for frame no. 426.*

Next we visualize the aggregated estimated network. As the network is quantified by the function $w : \mathbb{R}^5 \to \mathbb{R}$ it is difficult to visualize w in a 2-dimensional medium. A shiny app visualizing w can be found in the supplemental material. Here we consider various time and space aggregated measures of propagation effects some of which are inspired by analogous concepts from graph theory.

In Figure 3 below we plot the the fitted version of the two bivariate functions $w^+, w^- : \mathbb{R}^2 \to \mathbb{R}$ given by

$$w^+(x', y') := \frac{1}{\deg^+(x', y')} \int_{\mathcal{S}} \int_{-\tau}^{0} |w(x, y, x', y', t)| dt dx dy \tag{3.1}$$

$$w^-(x, y) := \frac{1}{\deg^-(x, y)} \int_{\mathcal{S}} \int_{-\tau}^{0} |w(x, y, x', y', t)| dt dx' dy', \tag{3.2}$$

where

$$\deg^+(x', y') := \int_{\mathcal{S}} \int_{-\tau}^{0} 1_{(w(x,y,x',y',t)\neq 0)} dt dx dy$$

$$\deg^-(x, y) := \int_{\mathcal{S}} \int_{-\tau}^{0} 1_{(w(x,y,x',y',t)\neq 0)} dt dx' dy',$$

are the aggregated non-zero effects going out from (x', y') respectively in to (x, y). Thus w^+ quantifies the effect of (x', y') on *all* other coordinates relative to the aggregated non-zero effects, that is time and space aggregated

propagation effects *from* any (x', y'). Similarly w^- quantifies time and space aggregated propagation effects *to* any (x, y).

Next in Figure 4 (left) we plot the bivariate function,

$$w^{+-}(d, r) = \int_{\mathcal{S}\times\mathcal{S}} 1_{(\|(x,y)-(x',y')\|=d)} |w(x, y, x', y', r)| dx dy dx' dy'$$

giving the aggregation of in- and out effects as a function of spatial separation (distance between coordinates) and temporal separation (time delay).

Finally, Figure 5 shows a density plot of the estimated weight values in $\hat{w}$. The density plot is truncated as most weights are estimated to zero.

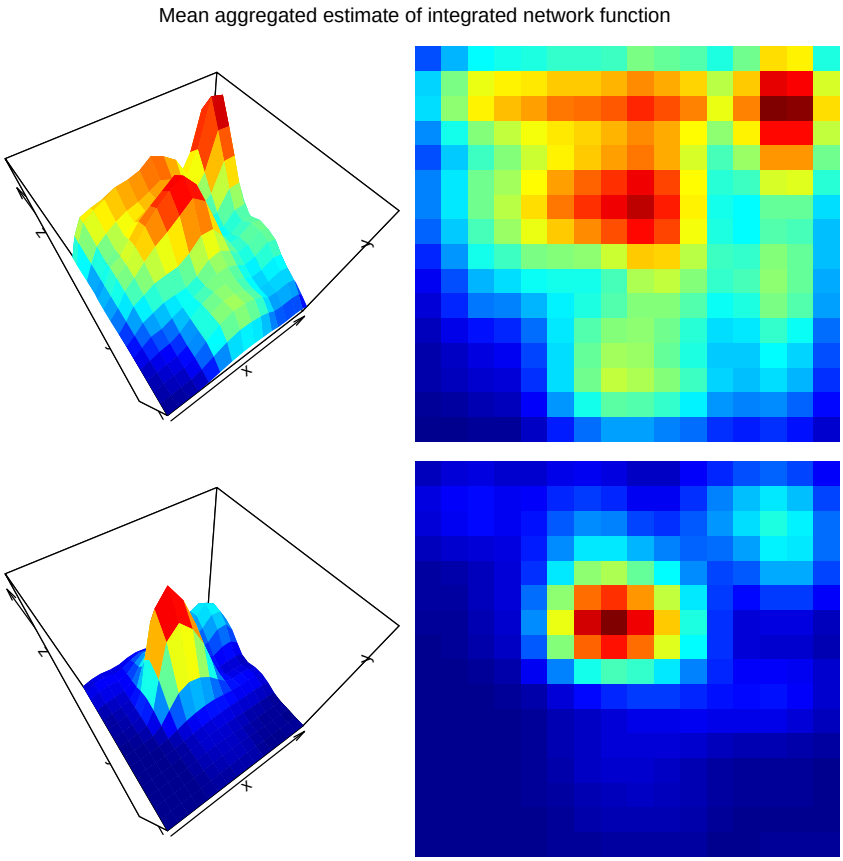


FIG 3. *The aggregated weight functions w^- (top) and w^+ (botom).*

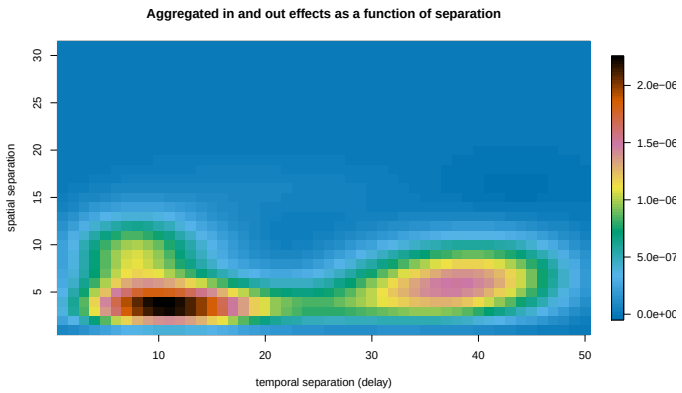


FIG 4. *The aggregated in and out effects as a function of temporal and spatial separation.*

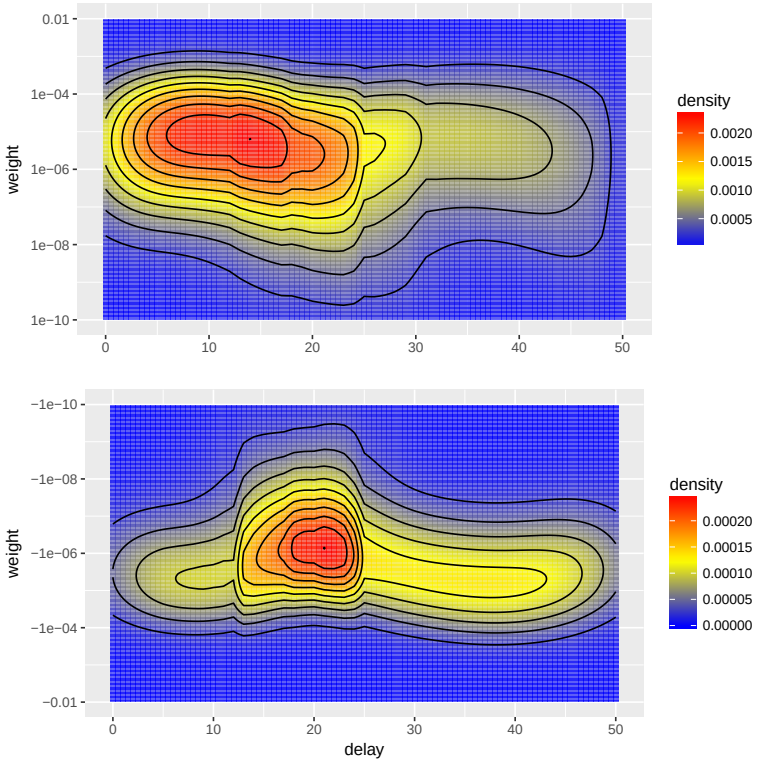


FIG 5. *Truncated density plots of the estimated weight values.*

From bottom panel in Figure 2 we see that an area is identified which across all pixels has a relatively great weight on every other pixel across all trials. Thus this area is identified by the model as important in the propagation of neuronal activity across all trials. We notice that this high output are as quantified by $\hat{w}$ overlaps with the strongest of the two high stimulus areas (CFOV) in Figure 1. This in turn suggests that the area receiving the strongest stimulus is also the area mostly responsible for propagating this input over time and space.

From the top panel we see that the pixels receiving propagation effects on the other hand is more scattered around the cortex. However the high input areas overlap with both of the high stimulus areas (CFOVs) in Figure 1 suggesting the existence of a propagation network connecting the high stimulus area and the low stimulus area and the immediate surroundings of the high stimulus area. We also notice an area in the top right corner of the frame that apparently also plays a role in the propagation of signals. The interpretation of this area is not immediately clear.

Figure 4 summarizes the network function as a function of temporal and spatial separation. The largest effects seem to occur with a delay of around 9 or 10 frames and affecting pixels in the immediate neighborhood the most (spatial distance 2 to 4 using the Manhattan metric). Especially the propagation effects do not seem to extend beyond 14 to 15 spatial units.

From Figure 5 we see inhibitory effects (high density area of negative weight values) seem to be delayed compared to excitatory effects (high density area of positive weight values).

3.2. *Simulation study of the fitted model.* We end this section by assessing the model fit in two different ways; how the model fits the observed data and how new data generated from the fitted model looks.

Next we examine if the model can generate data exhibiting characteristics similar to those observed in the real data set. We do that by simulating from the aggregated fitted (time and space discretized) model, see (4.3) in Proposition 4.1 below, using the aggregated fit discussed above.

Figure 6 gives a summary of the simulated data. Especially, the left plot in Figure 6 is a plot of the simulated time series for all pixels ignoring the spatial organization. This plot gives a rough impression of how the time evolution in the simulated data sets compares to that of the real data. In the right plot the simulated data for a specific time point is displayed to give a rough impression of the spatial organization of the simulated data. A movie visualizing the simulated data is provided as supplemental material.

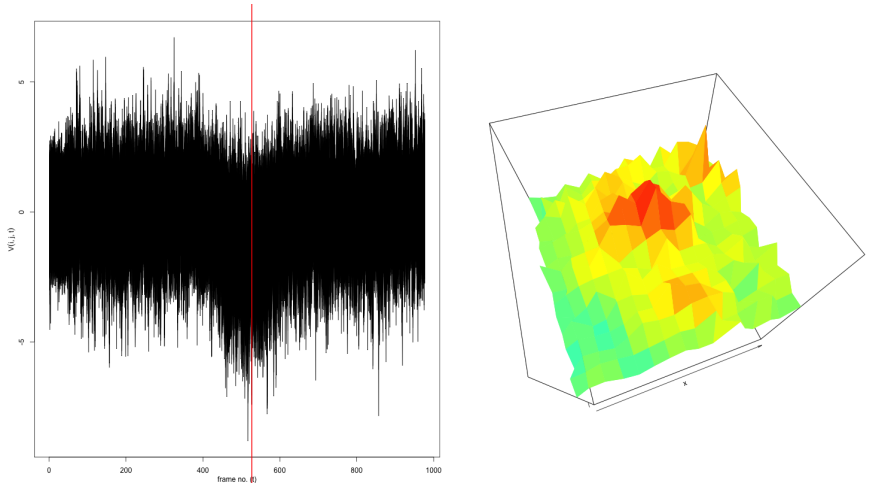


FIG 6. *The left plot shows the trajectories plotted for all pixels. The red vertical lines indicates the frame displayed in the right plot.*

We see that the simulated data shows temporary transient activity, namely the depolarization observed from around frame 400, similar to the real data, see Figure 1 above. Furthermore this activity is organized in a way spatially similar to that of the observed data set. These observations suggests that the dynamical model is to some extend capable of generating spatio-temporal data exhibiting characteristics shared by the real data.

4. A linear array model. The statistical model underlying the inferential framework used to obtain the results above in section 3 is based on a discretized version of the SFDE (2.3). The first step in our approach is to discretize space by aggregating the field over small areas. Let $(\mathcal{S}_{m,n})_{m,n} := (\mathcal{X}_m \times \mathcal{Y}_n)_{m=1,n=1}^{N_x,N_y}$ denote a partition of $\mathcal{S}$ with $D := N_x N_y$ elements with size $Leb(\mathcal{S}_{m,n}) = \Delta_s > 0$ for all m, n. We have from the integral representation of V in (2.7) that

$$\begin{aligned}\int_{\mathcal{S}_{m,n}} V(x,y,t)dxdy = \int_0^t \Bigg(& \int_{\mathcal{S}_{m,n}} s(x,y,u)dxdy \\ & + \int_{-\tau}^0 \sum_{i,j} \int_{\mathcal{S}_{i,j}} V(x',y',u+r) \int_{\mathcal{S}_{m,n}} w(x,y,x',y',r)dxdydx'dy'dr \\ & + \int_{\mathcal{S}_{m,n}} V(x,y,u)h(x,y)dxdy \Bigg) du + \int_{\mathcal{S}_{m,n}} \int_0^t \mathrm{d}W(u,x,y)dxdy. \end{aligned} \tag{4.1}$$

Here as noted in [23] the last term for each (x, y) is an Ito-integral with respect to a real valued Wiener process. Furthermore, using the covariance function for the random field from (2.4), the covariance of two such terms is

$$\begin{aligned} E\Big(\int_{\mathcal{S}_{m,n}}\int_0^t \mathrm{d}W(u,x,y)dxdy \int_{\mathcal{S}_{i,j}}\int_0^t \mathrm{d}W(u,x,y)dxdy'\Big) \\ = \int_{\mathcal{S}_{m,n}}\int_{\mathcal{S}_{i,j}} E(W(t,x,y)W(t,x',y'))dx'dy'dxdy \\ = \int_{\mathcal{S}_{m,n}}\int_{\mathcal{S}_{i,j}} tc(x-x',y-y')dx'dy'dxdy. \end{aligned}$$

We then apply a Riemann type approximation of the space integrals over the partition sets $\mathcal{S}_{m,n}$ on the left and the right of (4.1). This leads us to consider the D-dimensional real valued stochastic process $\tilde{V}$ with the (m,n)th coordinate process given by

$$\begin{aligned} \tilde{V}((m,n),t) = \int_0^t \Big(& \tilde{S}(u)(x_m,y_n) \\ & + \int_{-\tau}^0 \sum_{i,j} \tilde{V}((i,j),u+r) \int_{\mathcal{S}_{m,n}} w(x,y,x_i,y_j,r)dxdydr \\ & + \tilde{H}(V(u))(x_m,y_n)\Big)du + \int_0^t \mathrm{d}\tilde{W}((m,n),u)du \end{aligned} \tag{4.2}$$

as a space discretized model for the random field V. Here $\tilde{\boldsymbol{W}} := (\tilde{W}(t))_t$ is a D-dimensional Brownian motion on $(\Omega, \mathcal{F}, \mathrm{P})$ adapted to $(\mathcal{F}_t)_t$ with covariance matrix $\tilde{C}$ having rows given by

$$\tilde{C}_{(m,n)} := (c(x_m - x_1, y_n - y_1), \ldots, c(x_m - x_{N_x}, y_n - y_{N_y}))\Delta_s^2,$$

for each $(m,n) \in \{1,\ldots,N_x\} \times \{1,\ldots,N_y\}$. Furthermore, $\tilde{\mathrm{S}} : \mathcal{T} \to \mathbb{R}^D$ is given by $\tilde{\mathrm{S}}(u)(x_m,y_n) := \Delta_s s(x_m,y_n,u)$ and $\tilde{\mathrm{H}} : \mathbb{R}^D \to \mathbb{R}^D$ is given by $\tilde{\mathrm{H}}(\tilde{V}(u))(x_m,y_n) := \Delta_s h(x_m,y_n)\tilde{V}(x_m,y_n,u)$.

Thus aggregating the field over the spatial partition $(\mathcal{S}_{m,n})$ leads to a multi-dimensional SDDE as an approximate model for our data. Such models are mathematically easier to handle compared to the random field model (2.7), see e.g. [21]. Especially, we can obtain a discrete time approximation to the solution (4.2) using an Euler scheme following [6].

PROPOSITION 4.1. *Let $(t_k)_{k=-L}^M$ with $\Delta_t := t_{k+1} - t_k > 0$ denote a partition of $\mathcal{T}$. With $(\tilde{w}_k)_{k=-L}^{-1}$ a sequence of weight matrices depending on*

the network function w, the solution $(\tilde{V}^{\Delta_t}(k))_k$ to the forward Euler scheme

$$\begin{aligned}&\tilde{V}^{\Delta_t}(k+1) - \tilde{V}^{\Delta_t}(k)\\ &\quad = \left(\tilde{S}(t_k) + \sum_{l=-L}^{-1} \tilde{w}_l \tilde{V}^{\Delta_t}(k+l) + \tilde{H}(\tilde{V}^{\Delta_t}(k))\right)\Delta_t + \tilde{C}\sqrt{\Delta_t}\epsilon_k,\end{aligned} \tag{4.3}$$

where $(\epsilon_k)_k$ are independent and $N(0, I_D)$ distributed, converges weakly to the solution $\tilde{V}$ from (4.2).

PROOF. Follows from [6], see Appendix A. □

Next as the component functions; the stimulus functions s, the network function w and the short range memory h, are assumed to be square integrable we can use basis expansions to represent them. This in turn allow us to formulate the time and space discretized model as a linear regression model.

PROPOSITION 4.2. *Let $p_s, p_w, p_h \in \mathbb{N}$, $p := p_s + p_w + p_h$ and*

$$s(x, y, t) = \sum_{q=1}^{p_s} \alpha_q \phi_q(x, y, t). \tag{4.4}$$

$$w(x, y, x', y', t) = \sum_{q=1}^{p_w} \beta_q \phi_q(x, y, x', y', t) \tag{4.5}$$

$$h(x, y) = \sum_{q=1}^{p_h} \gamma_q \phi_q(x, y). \tag{4.6}$$

for three sets of basis coefficients $(\alpha_q)_q$, $(\beta_q)_q$ and $(\gamma_q)_q$. We can then write the dynamical model (4.3) as a linear regression model

$$y_k = X_k\theta + e_k, \quad k = 0, \ldots, M-1 \tag{4.7}$$

where (e_k) is i.i.d. $N_D(0, \Sigma)$ with $\Sigma := \tilde{C}^\top \tilde{C}\Delta_t$ and

$$y_k := \tilde{V}^{\Delta_t}(k+1), \quad X_k := (S_k \mid F_k \mid H_k), \quad \theta := \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}, \tag{4.8}$$

for each $k = 0, \ldots, M-1$ *are* $D\times 1$, $D\times p$ *and* $p\times 1$ *respectively . Furthermore the component matrices are*

(4.9)
$$S_k := \begin{pmatrix} \phi_1(x_1, y_1, t_k) & \cdots & \phi_{p_s}(x_1, y_1, t_k) \\ \vdots & \vdots & \vdots \\ \phi_1(x_{N_x}, y_{N_y}, t_k) & \cdots & \phi_{p_s}(x_{N_x}, y_{N_y}, t_k) \end{pmatrix}, F_k := \begin{pmatrix} F_k^{(1,1)} \\ \vdots \\ F_k^{(N_x,N_y)} \end{pmatrix},$$

with $F_k^{(m,n)}$ *a* $1 \times p_w$*-vector with entries*

(4.10)
$$F_{k,q}^{(m,n)} := \sum_{l=-L}^{-1} \sum_{i,j} \tilde{V}^{\Delta_t}((i,j), k+l) \int_{t_l}^{t_{l+1}} \int_{\mathcal{S}_{m,n}} \phi_q(x, y, x_i, y_j, r) dx dy dr$$

for $q \in \{1, \ldots, p_w\}$*, and*

(4.11)
$$H_k := \begin{pmatrix} \phi_1(x_1, y_1)\tilde{V}^{\Delta_t}((1,1), k) & \cdots & \phi_{p_h}(x_1, y_1)\tilde{V}^{\Delta_t}((1,1), k) \\ \vdots & \vdots & \vdots \\ \phi_1(x_{N_x}, y_{N_y})\tilde{V}^{\Delta_t}((N_x, N_y), k) & \cdots & \phi_{p_h}(x_{N_x}, y_{N_y})\tilde{V}^{\Delta_t}((N_x, N_y), k) \end{pmatrix}.$$

Finally defining

$$y := \begin{pmatrix} y_1 \\ \vdots \\ y_M \end{pmatrix}, \quad X := \begin{pmatrix} X_1 \\ \vdots \\ X_M \end{pmatrix}, \tag{4.12}$$

the associated negative log-likelihood can be written as

$$l(\theta, \Sigma) = \frac{M}{2} \log |\Sigma| + \|I_M \otimes \Sigma^{-1/2}(y - X\theta)\|_2^2, \tag{4.13}$$

with I_M *the* $M \times M$ *identity matrix.*

Proof. See Appendix A. □

We note that due to the structure of the linear model (4.7) the MLEs of θ and Σ are not available in closed form. Especially the normal equations characterizing the MLEs are coupled in this model leading to a generalized least squares type estimation problem.

Next we will discuss an approach that will allow us to obtain sparse estimates of θ as well as the covariance Σ.

4.1. *Penalized linear array model.* In order to obtain time and space localized estimates of the component functions, we will maximize a regularized version of (4.13). This is achieved by solving the unconstrained problem

$$\min_{\theta\in\mathbb{R}^p,\Sigma\in M^{D\times D}} l_X(\theta,\Sigma)+\lambda_1 J_1(\theta)+\lambda_2 J_2(\Sigma), \tag{4.14}$$

where J_1 and J_2 are convex penalty functions and $\lambda_1 \geq 0$ and $\lambda_2 \geq 0$ the penalty parameters controlling the amount of penalization or regularization. For $\lambda_i > 0$ and J_i, $i = 1, 2$ non-differentiable penalty functions, solving (4.14) results in a sparse estimates. In the following we will use the lasso or ℓ_1 penalty i.e. $J_1 = J_2 = \|\cdot\|_1$, see [30].

Following [27] we solve (4.14) using their approximate MRCE algorithm. In our setup the steps are as follows:

1. For $\Sigma = I_D$ and a penalty parameters $\lambda_1 \geq \ldots \geq \lambda_K$, $K \in \mathbb{N}$. solve

$$\min_{\theta\in\mathbb{R}^p} l_X(\theta,\Sigma)+\lambda_1 J_1(\theta), \tag{4.15}$$

2. For $i \in \{1,\ldots,K\}$ let $\hat{\theta}_{\lambda_i}$ denote the estimate from step 1. Solve

$$\hat{\Sigma} := \arg\min_{\Sigma} tr(\hat{\Sigma}_R\Sigma) - \log|\Sigma| + \lambda_2 J_2(\Sigma)$$

   with $\hat{\Sigma}_R := (y - X\hat{\theta}_{\lambda_i})^\top (y - X\hat{\theta}_{\lambda_i})$, using graphical lasso, see [17].
3. Repeat step 1. with data, $\tilde{y} := \hat{\Sigma}^{-1/2}y$ and $\tilde{X} := \hat{\Sigma}^{-1/2}X$.

Solving the problem (4.14) requires a numerical procedure. However from a computationally viewpoint evaluating l given in (4.13) becomes infeasible as the design matrix X in practice is unwieldily. For instance for each of the trials considered in section 3 the design X takes up around 70GB of memory depending on the number of basis functions used. In addition even if we could allocate the memory needed to store X the time needed to compute the entries in X would be considerable.

However, using tensor product basis functions (see Appendix B) we can, by exploiting the rotated H transform ρ from [11], avoid forming the potentially enormous design matrix X.

DEFINITION 4.1. *For A an $p_1 \times \cdots \times p_d$ array and X_i $n_i \times p_i$ for $i \in \{1,\ldots,d\}$ the rotated H transform ρ is defined as the map such that*

$$X_d \otimes \cdots \otimes X_1 \mathrm{vec}(A) = \mathrm{vec}(\rho(X_d, \rho(\ldots, \rho(X_1, A)))), \tag{4.16}$$

where vec *is the vectorization operator.*

The above definition is not very enlightening in terms of how ρ actually computes the matrix vector product. These details can be found in [11]. Given Definition 4.1 however, we see that for a tensor structured matrix we can compute matrix vector products via ρ using only its tensor components. This makes ρ very memory efficient. Furthermore, as discussed in [14], [7], [11], ρ is also computationally efficient as the left hand side of (4.16) takes more multiplications to compute than the right hand side. In this light the following proposition is relevant.

PROPOSITION 4.3. *With $p_x, p_y, p_l, p_t \in \mathbb{N}$, $p_s = p_x p_y p_t$, $p_w := p_x p_y p_x p_y p_l$, $p_h := p_x p_y$, using the tensor product construction, we can write*

$$
\begin{aligned}
s(x,y,t) &= \sum_{\boldsymbol{j}=(j_1,j_2,j_3)}^{p_s} \alpha_{\boldsymbol{j}} \phi^x_{j_1}(x)\phi^y_{j_2}(y)\phi^t_{j_3}(t) \\
w(x,y,x',y',t) &= \sum_{\boldsymbol{j}=(j_1,j_2,j_3,j4_5)}^{p_w} \beta_{\boldsymbol{j}} \phi^x_{j_1}(x)\phi^y_{j_2}(y)\phi^x_{j_3}(x')\phi^y_{j_4}(y')\phi^l_{j_5}(t) \\
h(x,y) &= \sum_{\boldsymbol{j}=(j_1,j_2)}^{p_h} \gamma_{\boldsymbol{j}} \phi^x_{j_1}(x)\phi^y_{j_2}(y).
\end{aligned}
$$

Let A be a $p_x \times p_y \times p_t$ array, B a $p_x \times p_y \times p_x p_y p_l$ array and Γ a $p_x \times p_y$ matrix containing the basis coefficients α, β and γ resepctively. For appropriately defined matrices $\phi^x, \phi^y, \phi^t, \Phi^{xyt}, \Phi^x, \Phi^y$

$$
X\theta = \operatorname{vec}(\rho(\phi^t, \rho(\phi^y, \rho(\phi^x, A))) + \rho(\Phi^{xyt}, \rho(\Phi^y, \rho(\Phi^x, B))) \tag{4.17}
$$

$$
+ v_{-1,M-1} \odot C) \tag{4.18}
$$

with C a $N_x \times N_y \times M$ array and $\odot$ denoting the Hadamard (entry-wise) product.

PROOF. See Appendix A. □

Proposition 4.3 shows that we can in fact write the linear model (4.7) as a three component (partially) 3 dimensional linear array model, see [20] for multi component array models. Especially we can completely avoid forming the component design matrices S, F and H when computing the matrix vector product involving the potentially enormous design matrix. This in turn implies that we can build a numerical procedure based on the gradient descent proximal gradient (gd-pg) algorithm proposed in [20], that will solve the non-differentiable penalized estimation problem. This algorithm uses a minimal amount of memory while exploiting the efficient array arithmetic. In Appendix C below additional details about the implementation is given.

5. Discussion. It was crucial for the methodology proposed in this paper that the numerical computations related to the linear array model can be implemented using the algorithm proposed in [20]. This allowed us to obtain a design matrix free procedure with a minimal memory footprint that can utilize the highly efficient array arithmetic, see [14], [7] and [11]. Consequently we were able to fit the model to VSD imaging data with hundreds of pixels in the spatial dimension and recorded over thousands of time points while modeling very long time delays, on a standard laptop computer in a matter of minutes. Given the results in [20] we expect our procedure to scale well with the size of the data.

We note that the large-scale setup considered in this paper is computationally prohibitive using conventional time series techniques. Especially, econometric approaches like vector autoregressions (VAR) usually consider time series with no more than 10 observations in the spatial dimension and few lags as the number of parameters grows quadratically in the size of the spatial dimension, see [15]. In [13] an approach to high dimensional VAR analysis is given and in [32] a VAR model is applied to fMRI brain image data. However, the size of the data sets considered in these papers is much smaller than what we considered here. Furthermore, aside from the computational aspect, the enormous number of parameters fitted when employing a VAR model in a large-scale setting would be quite difficult to interpret. Within our framework, by modelling the array data as a random field model, we instead estimate smooth functions that in turn are easier to interpret.

APPENDIX A: PROOFS

PROOF OF PROPOSITION 1. With w the network function let $\tilde{w}$ denote a $D \times D$ matrix-valued signed measure on $[-\tau, 0]$ with density $\tilde{F} : \mathbb{R} \to \mathbb{R}^{D\times D}$,

$$\tilde{F}(t) := \begin{pmatrix} \int_{\mathcal{S}_{1,1}} w(x, y, x_i, y_j, t)dxdy & \dots & \int_{\mathcal{S}_{N_x,N_y}} w(x, y, x_i, y_j, t)dxdy \\ \vdots & \ddots & \vdots \\ \int_{\mathcal{S}_{1,1}} w(x, y, x_{N_x}, y_{N_y}, t)dxdy & \dots & \int_{\mathcal{S}_{N_x,N_y}} w(x, y, x_{N_x}, y_{N_y}, t)dxdy \end{pmatrix},$$

with respect to the Lebesgue measure on $\mathbb{R}$. Following [6] we then consider the stochastic delay differential equation

$$\mathrm{d}\tilde{V}(t) = (\tilde{S}(t) + \int_{-\tau}^{0} \tilde{w}(\mathrm{d}r)\tilde{V}(t + r) + \tilde{H}(\tilde{V}(t)))\mathrm{d}t + \tilde{C}\mathrm{d}\tilde{W}(t) \tag{A.1}$$

$$\tilde{V}(0) \in \mathbb{R}^D, \quad \tilde{V}(u) = \tilde{V}_0(u) \quad u \in (-\tau, 0), \tag{A.2}$$

where $\tilde{V}(0)$ and $\tilde{V}_0 \in C([-\tau, 0], \mathbb{R}^D)$ are initial conditions. A solution to (A.1) and (A.2) is then given by the integral equation (4.2) describing the

coordinate-wise evolution in the space discretized model for the random field V.

The sequence of $D \times D$ weight matrices $(\tilde{w}_k)_k$ is then defined by

$$\tilde{w}_k := \int_{-\tau}^{0} \mathbf{1}_{[t_k,t_{k+1})}(s)\mathrm{d}\tilde{w}(s) = \int_{-\tau}^{0} \mathbf{1}_{[t_k,t_{k+1})}(s)\tilde{F}(s)\mathrm{d}s,$$

where $\mathbf{1}_{[t_k,t_{k+1})}$ is equal to I_D on $[t_k, t_{k+1})$ and zero otherwise. Especially the entry in the (m,n)th row and (i,j)th column of each $\tilde{w}_k$ is given as

$$\tilde{w}_k((m,n),(i,j)) = \int_{t_k}^{t_{k+1}} \int_{\mathcal{S}_{m,n}} w(x,y,x_i,y_j,s)dxdy\mathrm{d}s. \tag{A.3}$$

Letting $(\epsilon_k)_k$ denote a sequence of iid $N(0, I_D)$ variables the Euler scheme from [6] is then given by the D-dimensional discrete time process $(\tilde{V}^{\Delta_t}(k))_k$ solving the stochastic difference equation (4.3) for $k = 0, \ldots, M-1$ with initial conditions $\tilde{V}^{\Delta_t}(l) = \tilde{V}_0(t_l)$ for $l = -L, \ldots, -1$ and $\tilde{V}^{\Delta_t}(0) = \tilde{V}(0)$. Thus by Theorem 1.2 in [6] and using that the deterministic function s is continuous, the solution $(\tilde{V}^{\Delta_t}(k))_k$ to (4.3) converges weakly to the solution process $(\tilde{V}(t))_t$, defined in (4.2), that solves the SDDE (A.1) and (A.2) . □

Proof of Proposition 2. First using the expansion in (4.5) we can write the entries in each weight matrix $\tilde{w}_k$ from (A.3) as

$$\begin{aligned}\tilde{w}_k((m,n),(i,j)) &= \int_{t_k}^{t_{k+1}} \int_{\mathcal{S}_{m,n}} w(x,y,x_i,y_j,s)dxdy\mathrm{d}s \\ &= \sum_q \beta_q \int_{t_k}^{t_{k+1}} \int_{\mathcal{S}_{m,n}} \phi_q(x,y,x_i,y_j,s)dxdy\mathrm{d}s.\end{aligned}$$

Then we can write the (m,n)th coordinate of (4.3) as

$$\begin{aligned}\tilde{V}^{\Delta_t}((m,n),k+1) &= \Big(\tilde{s}(x_m,y_n,t_k) + \sum_{l=-L}^{-1} \tilde{w}_l((m,n))\tilde{V}^{\Delta_t}(k+l) \\ &\quad + \tilde{h}(x_m,y_n)\tilde{V}^{\Delta_t}((m,n),k)\Big)\Delta_t + \tilde{C}_{(m,n)}\sqrt{\Delta_t}\epsilon_k \\ &= \Delta_t \sum_q \alpha_q \phi_q(x_m,y_n,t_k) + \beta_q F_{k,q}^{m,n} + \gamma_q \phi_q(x_m,y_n)\tilde{V}^{\Delta_t}((m,n),k) \\ &\quad + \tilde{C}_{(m,n)}\sqrt{\Delta_t}\epsilon_k \end{aligned} \tag{A.4}$$

with $F_k^{(m,n)}$ a $1 \times p_w$-vector with entries given by (4.10). Defining y_k, X_k, θ and e_k as in (4.8) the model equation (4.7) follows from (A.4).

We then have the following transition density for the model

$$f(y_k \mid X_k) = (\sqrt{2\pi})^{-D}|\Sigma|^{-1/2}\exp(-(y_k - X_k\theta)^\top\Sigma^{-1}(y_k - X_k\theta)/2).$$

As $y_k \mid X_k$ is independent of $y_0, \dots y_{k-1}, X_0, \dots X_{k-1}$ we get by successive conditioning that we can write the joint conditional density as

$$\begin{aligned} &f(y_0, \dots, y_{M-1} \mid X_0, \dots, X_{M-1}) \\ &\qquad = (\sqrt{2\pi})^{-MD}|\Sigma|^{-M/2}\exp\Big(-\frac{1}{2}\sum_{k=0}^{M-1}(y_k - X_k\theta)^\top\Sigma^{-1}(y_k - X_k\theta)\Big). \end{aligned}$$

Taking minus the logarithm yields the negative log-likelihood

$$l(\theta, \Sigma) := \frac{M}{2}\log|\Sigma| + \frac{1}{2}\sum_{k=0}^{M-1}(y_k - X_k\theta)^\top\Sigma^{-1}(y_k - X_k\theta).$$

With X and y as in (4.12) it follows that

$$I_M \otimes \Sigma^{-1/2}(y - X\theta) = \begin{pmatrix} \Sigma^{-1/2}(y_1 - X_1\theta) \\ \vdots \\ \Sigma^{-1/2}(y_M - X_M\theta) \end{pmatrix} = \begin{pmatrix} \sum_j^D \Sigma_{1,j}^{-1/2}(y_1 - X_1\theta)_j \\ \vdots \\ \sum_j^D \Sigma_{D,j}^{-1/2}(y_1 - X_1\theta)_j \\ \vdots \\ \sum_j^D \Sigma_{D,j}^{-1/2}(y_D - X_D\theta)_j \end{pmatrix}$$

hence

$$\begin{aligned} \|I_M \otimes \Sigma^{-1/2}(y - X\theta)\|_2^2 &= \sum_k^M \sum_i^D \Big(\sum_j^D \Sigma_{i,j}^{-1/2}(y_k - X_k\theta)_j\Big)^2 \\ &= \sum_{k=0}^{M-1} \|\Sigma^{-1/2}(y_k - X_k\theta)\|_2^2 \\ &= \sum_{k=0}^{M-1}(y_k - X_k\theta)^\top\Sigma^{-1}(y_k - X_k\theta) \end{aligned}$$

yielding the expression for the negative log-likelihood given in (4.13). □

Proof of Proposition 3. Writing

$$(S \mid F \mid H) := \begin{pmatrix} S_1 & F_1 & H_1 \\ \vdots & \vdots & \vdots \\ S_M & F_M & H_M \end{pmatrix} = X$$

the claim follows if we show that S and F are appropriate 3-tensor matrices and that $H\gamma = V_{-1} \odot C$ for an appropriately defined array C.

Letting $\phi^x := (\phi_q(x_i))_{i,q}$ denote a $N_x \times p_x$ matrix with p_x basis functions evaluated at N_x points in the x domain it follows directly from the definition of the tensor product that we can write $S = \phi^x \otimes \phi^y \otimes \phi^t$.

Next let $\Phi^x := (\int_{\mathcal{X}_i} \phi_q(x))_{i,q}$ denote the integrated version of ϕ^x. Then inserting the tensor basis functions in to (4.10) we can write the entries in $F_k^{m,n}$ as

$$\begin{aligned} F_{k,\boldsymbol{q}}^{m,n} &= \sum_{l=-L}^{-1} \sum_{i,j} \tilde{V}_{k+l}^{\Delta_t}(x_i, y_j) \int_{t_l}^{t_{l+1}} \int_{\mathcal{S}_{m,n}} \phi_{q_1}(x)\phi_{q_2}(y)\phi_{q_3}(x_i)\phi_{q_4}(y_j)\phi_{q_5}(s)dxdyds \\ &= \int_{\mathcal{X}_m} \phi_{q_1}(x)dx \int_{\mathcal{Y}_n} \phi_{q_2}(y)dy \sum_{l=-L}^{-1} \sum_{i,j} \tilde{V}_{k+l}^{\Delta_t}(x_i, y_j)\phi_{q_3}(x_i)\phi_{q_4}(y_j) \int_{t_l}^{t_{l+1}} \phi_{q_5}(s)\mathrm{d}s \end{aligned}$$

for $\boldsymbol{q} = (q_1, q_2, q_3, q_4, q_5)$. Letting Φ^{xyl} denote a $M \times p_x p_y p_l$ matrix with entries

$$\Phi_{k,\boldsymbol{q}}^{xyl} := \sum_{l=-L}^{-1} \sum_{i,j} \tilde{V}_{k+l}^{\Delta_t}(x_i, y_j)\phi_{q_3}(x_i)\phi_{q_4}(y_j) \int_{t_l}^{t_{l+1}} \phi_{q_5}(s)\mathrm{d}s \tag{A.5}$$

for $\boldsymbol{q} = (q_3, q_4, q_5)$ we can write F as

$$F = \Phi^{xyt} \otimes \Phi^y \otimes \Phi^x.$$

Finally with H_k given in (4.11) we can write

$$\begin{aligned} H_k\gamma &= \begin{pmatrix} V_k^{\Delta_t}(x_1, y_1) \sum_{\boldsymbol{q}} \phi_{q_1}^x(x_1)\phi_{q_2}^y(y_1)\gamma_{\boldsymbol{q}} \\ \vdots \\ V_k^{\Delta_t}(x_{N_x}, y_{N_y}) \sum_{\boldsymbol{q}} \phi_{q_1}^x(x_{N_x})\phi_{q_2}^y(y_{N_y})\gamma_{\boldsymbol{q}} \end{pmatrix} \\ &= V_k^{\Delta_t} \odot \phi^x \otimes \phi^y\gamma \\ &= V_k^{\Delta_t} \odot \mathrm{vec}(\phi^x\Gamma(\phi^y)^\top) \end{aligned}$$

Letting C be the $N_x \times N_y \times M$ array containing M copies of the $N_x \times N_y$ matrix $\phi^y\Gamma(\phi^x)^\top$ we can then write $H\gamma = V_{-1} \odot C$. □

APPENDIX B: TENSOR PRODUCT BASIS

Consider a d-variate function $f \in L^2(\mathbb{R}^d)$ that we want to represent using some basis expansion. Instead of directly specifying a basis for $L^2(\mathbb{R}^d)$ we will specify d marginal sets of univariate functions

$$(\phi_{1,k_1})_{k_1=1}^{\infty}, \ldots, (\phi_{d,k_d})_{k_d=1}^{\infty} \tag{B.1}$$

with each marginal set a basis for $L^2(\mathbb{R})$. Then for any $(k_1, \ldots, k_d) \in \mathbb{N}^d$ we may define a d-variate function $\pi_{k_1,\ldots,k_d} : \mathbb{R}^d \to \mathbb{R}$ via the product of the marginal functions i.e.

$$(x_1, \ldots, x_d) \mapsto \pi_{k_1,\ldots,k_d}(x_1, \ldots, x_d) := \prod_{i=1}^{d} \phi_{i,k_i}(x_i). \tag{B.2}$$

Since each marginal set of functions in (B.1) constitute a basis of $L^2(\mathbb{R})$ it follows using Fubini's theorem, that the induced set of d-variate functions $(\pi_{k_1,\ldots,k_d})_{k_1,\ldots,k_d}$ constitute a basis of $L^2(\mathbb{R}^d)$. Especially again by Fubini's theorem we note that if the functions in (B.1) all are orthonormal marginal bases then they generate an orthonormal basis of $L^2(\mathbb{R}^d)$. Finally, if the marginal functions have compact support then the induced d-variate set of functions will also have compact support.

APPENDIX C: IMPLEMENTATION DETAILS

Here we elaborate a little on various details pertaining the implementation of the inferential procedure for the specific data considered in section 3 and on some more general details relating to the computations.

C.1. Representing the drift components.

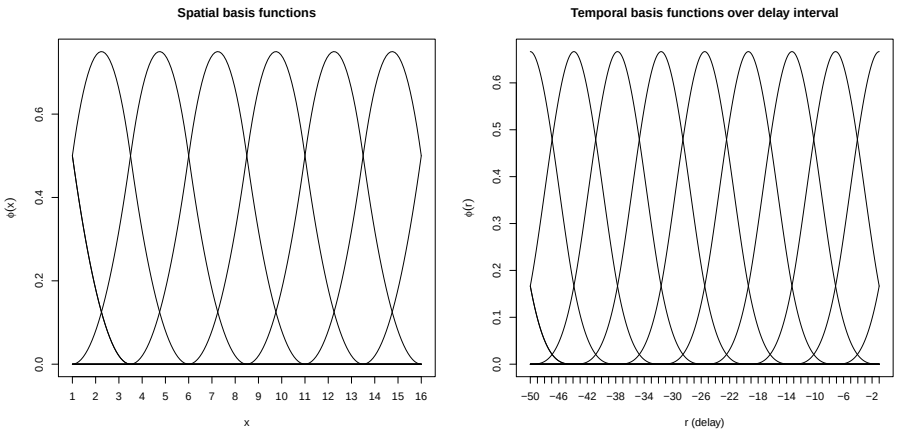


FIG 7. *The spatial basis splines (left) and the temporal basis splines (right).*

C.1.1. *Network function and short range function.* As explained in Appendix B we can use tensor basis functions to represent the component

functions. For the analysis of the brain image data we will use B-splines as basis functions in each dimension as these have compact support. Specifically in the spatial dimensions we will use quadratic B-splines while in the temporal dimension we will use cubic B-splines, see Figure 7 below.

C.1.2. *Stimulus function.* Using separable functions to represent the stimulus, as discussed in Appendix B, and exploiting that we know when the stimulus is presented we can pre-estimate the generic temporal factor, ϕ^t, of the stimulus function in Proposition 4.3. The underlying reasoning behind this is that the external input should arrive at the same time across the visual cortex, hence the stimulus function s should be temporally synchronized across space. However the amplitude of the stimulus functions should be estimated in the overall model fitting, especially setting s to zero in areas where no input is detected.

More concretely with s_l and s_r respectively indicating the stimulus start and the stimulus stop, for some smooth function g, we fit the model

$$V(x, y, t) = g(t) + \epsilon, \quad (x, y, t) \in \mathcal{S} \times [s_l, s_r].$$

Representing g in a tensor basis of cubic B-splines, we can do this in a fraction of second for a single trial, using the `glamasso` R-package, see [19].

Having obtained the estimate $\hat{g}$ we use $m := \arg\min_t \hat{g}(t)$ as an estimate of the synchronized peak input time and model ϕ^t using a scaled gamma density function with mode m and scale 20, see Figure 8.

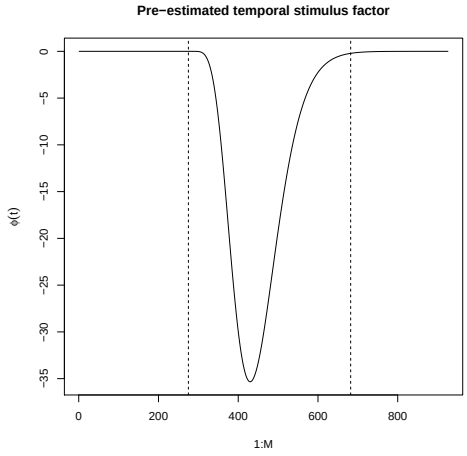


FIG 8. *The basis functions used to represent the stimulus response functions. The dotted vertical lines indicates the stimulus start (frame 325) and stop (frame 732).*

C.2. Regression set up. For the single trial VSD data we have a total of 250112 observations arranged in a $16 \times 16 \times 977$ grid. We model a delay of $L := 50$ frames corresponding to $\tau \approx 31$ ms which gives us $M := 926$ modeled time points. As can be gleaned in Figure 7 the number of basis functions in the spatial dimensions is $p_x = p_y := 8$ in each and $p_l := 11$ in the temporal dimension and $p_t := 1$. With this setup we have us a total of of $p := 45184$ model parameters that need to be estimated. For comparison we note that a corresponding VAR(L) model would have $LD^2 = 3276800$ model parameters .

For the lasso regression we use parameter weights to allow for differential penalization. Especially we penalize the stimulus and short range coefficient less (weight 1) that the network coefficients (weight 10)

We fit the model for a sequence of penalty parameters $\lambda_{max} := \lambda_1 > \ldots > \lambda_k =: \lambda_{min}$ $k := 30$. Here λ_{max} can be computed given data as the smallest value yielding a zero solution to (4.14).

C.3. Computational details. Finally, some comments pertaining the computations involved when implementing the estimation procedure.

As noted above we use is the gd-pg algorithm from [20]. However, to implement that algorithm in this setup you need to split up the computations according to components. Especially matrix-vector products for the stimulus and network components can be carried out using the rotated H-transform from definition 4.1. However for the short range component it is only possible to carry out the matrix vector products as "two tensor array arithmetic" plus an entrywise product, as indicated in (4.18). This especially pertains to the gradient computations, which due to the component structure, splits up in to $3^2 = 9$ block-components, 5 of which involves the the short range component.

We note however that the vast majority of computations performed by the inferential procedure involves the stimulus component and the network component where the tensor structure can be efficiently exploited.

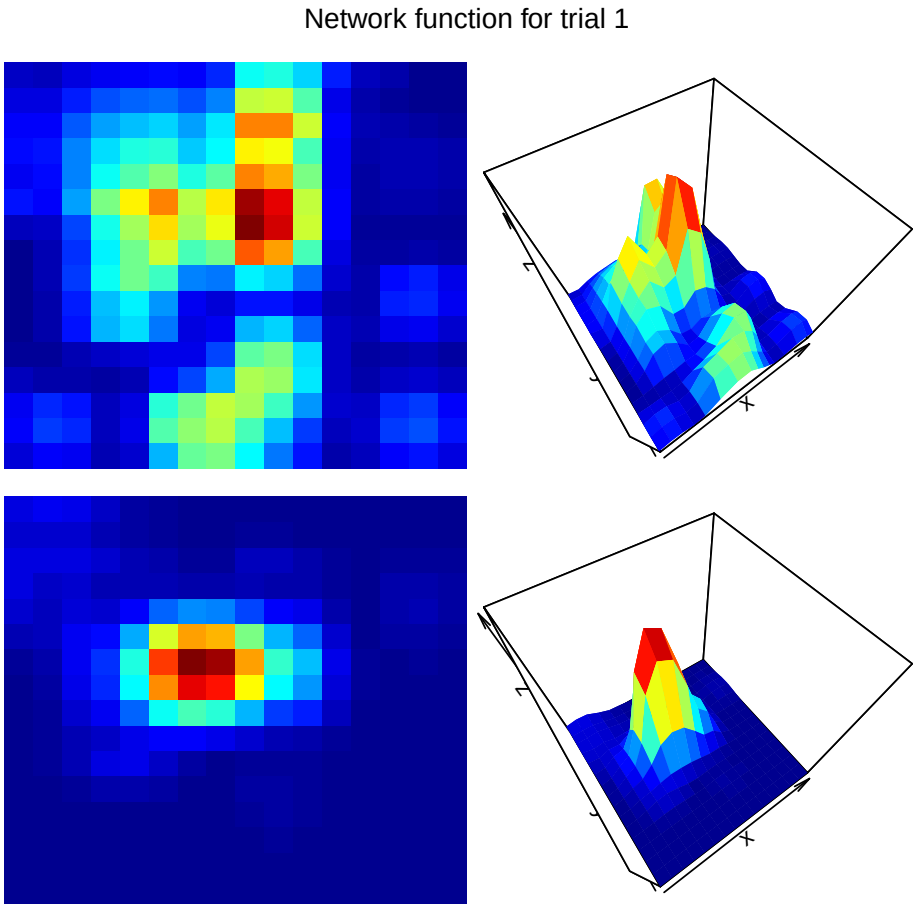
Finally for the network component one tensor, as shown in the proof of Proposition 4.3, corresponds to a convolution of the random field. This component has to be computed upfront which in principle can be very time consuming. However considering (A.5) this computation can be carried out using array arithmetic. Especially we can write the convolution tensor as

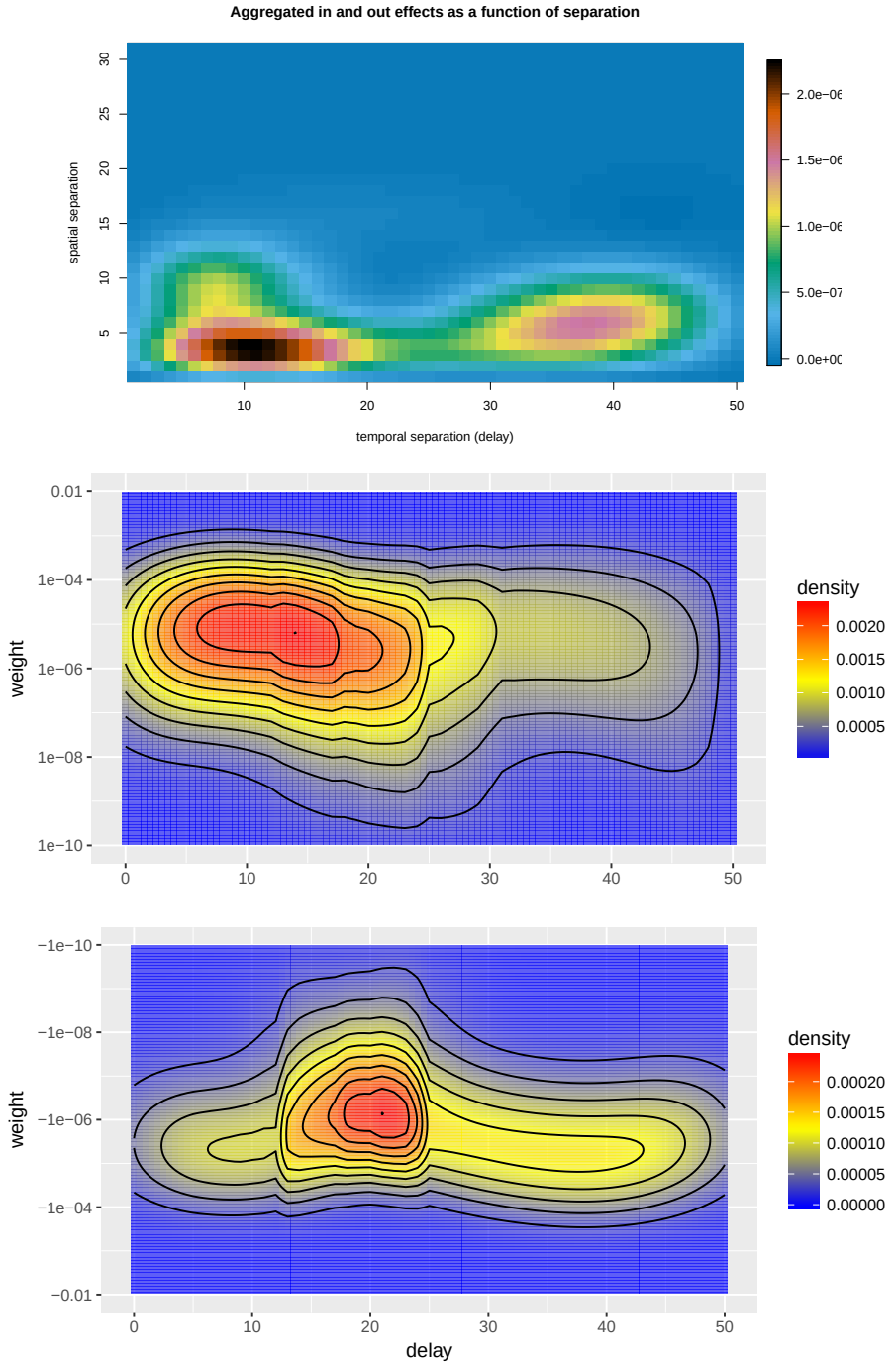
$$\Phi^{xyt} = \begin{pmatrix} \operatorname{vec}(\Phi_1^{xyl}) \\ \vdots \\ \operatorname{vec}(\Phi_M^{xyl}) \end{pmatrix}.$$

where Φ_k^{xyt} for each k is a $p_x \times p_y \times p_l$ array which according to (A.5) can be computed using the array arithmetic as

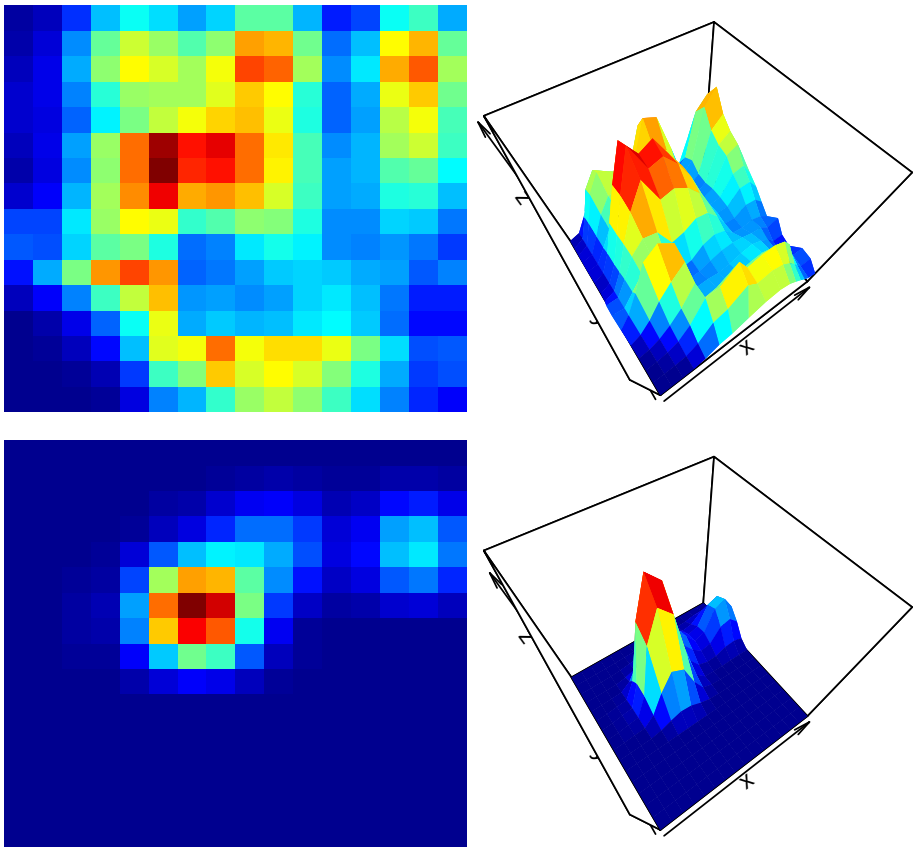
$$\Phi_k^{xyl} := \rho((\phi^l)^\top, \rho((\phi^y)^\top, \rho((\phi^x)^\top, v_{-L+k,k})).$$

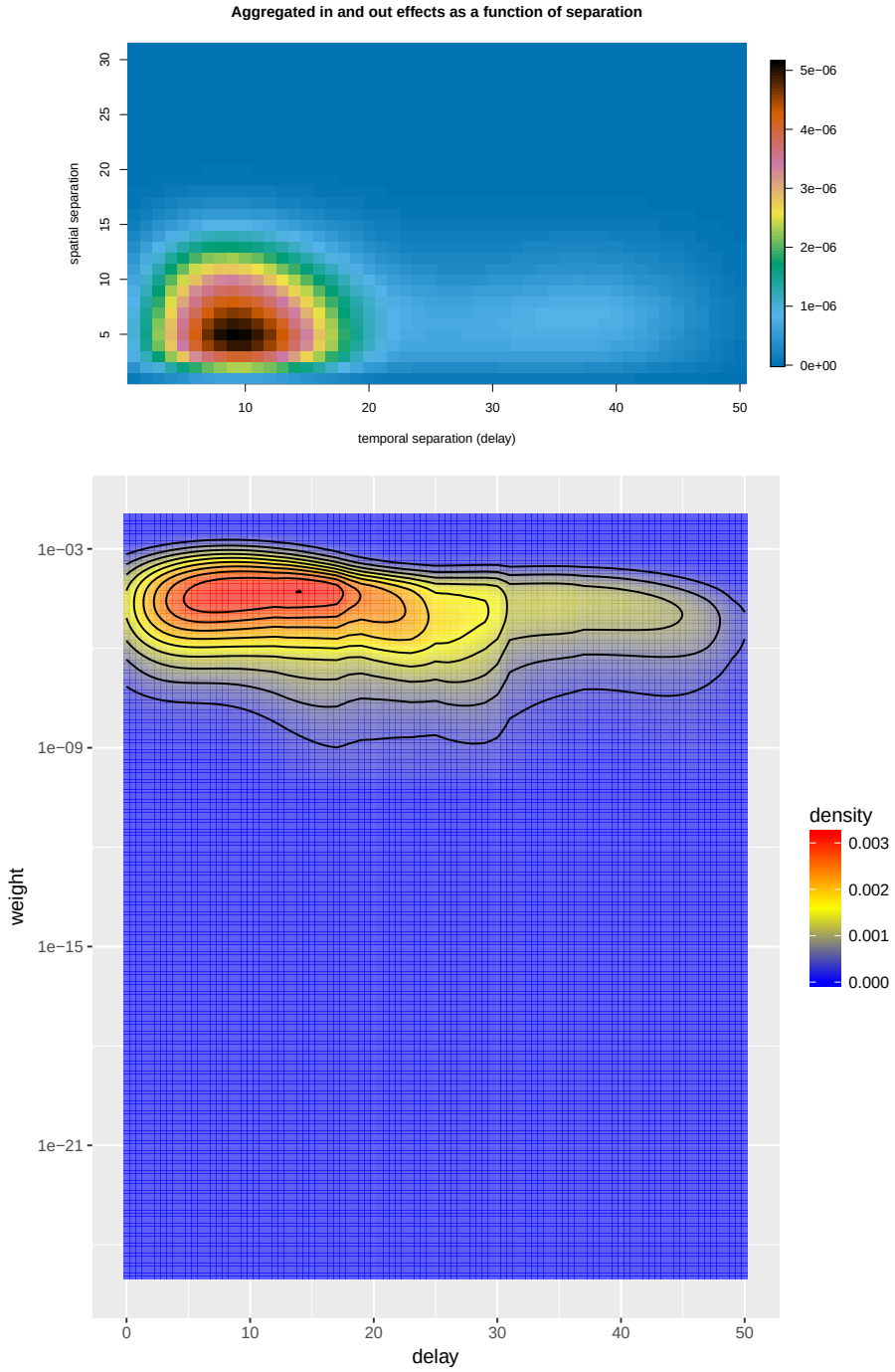
APPENDIX D: FIT FOR ALL TRIALS

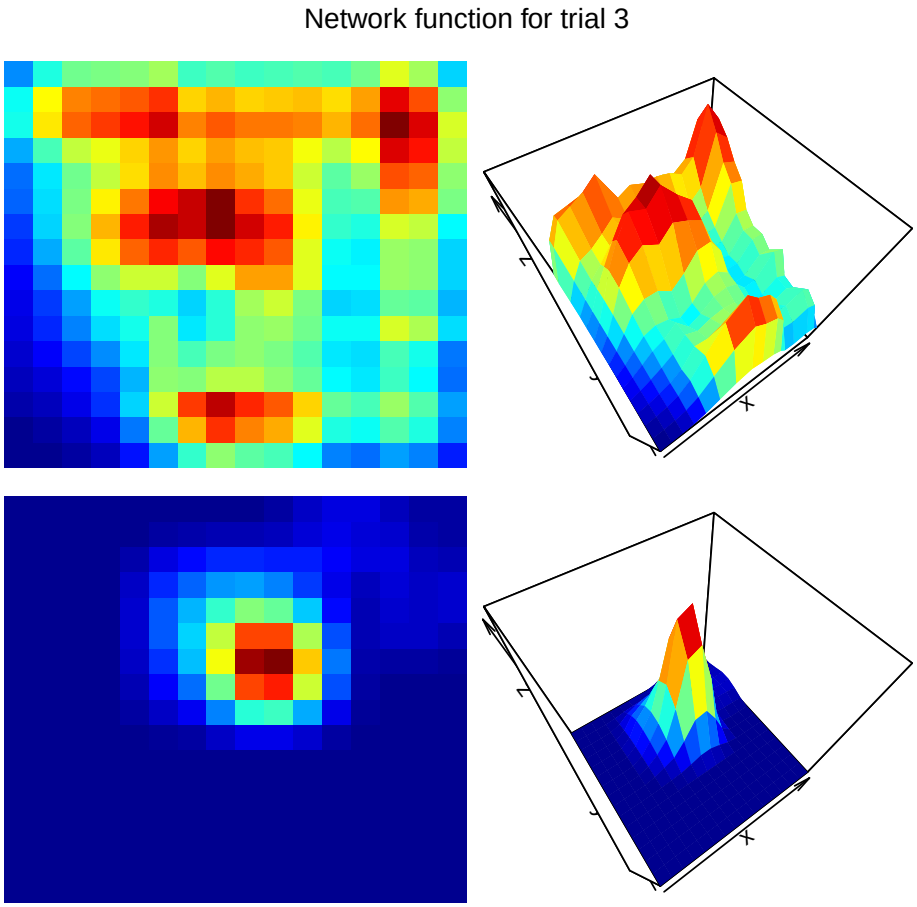


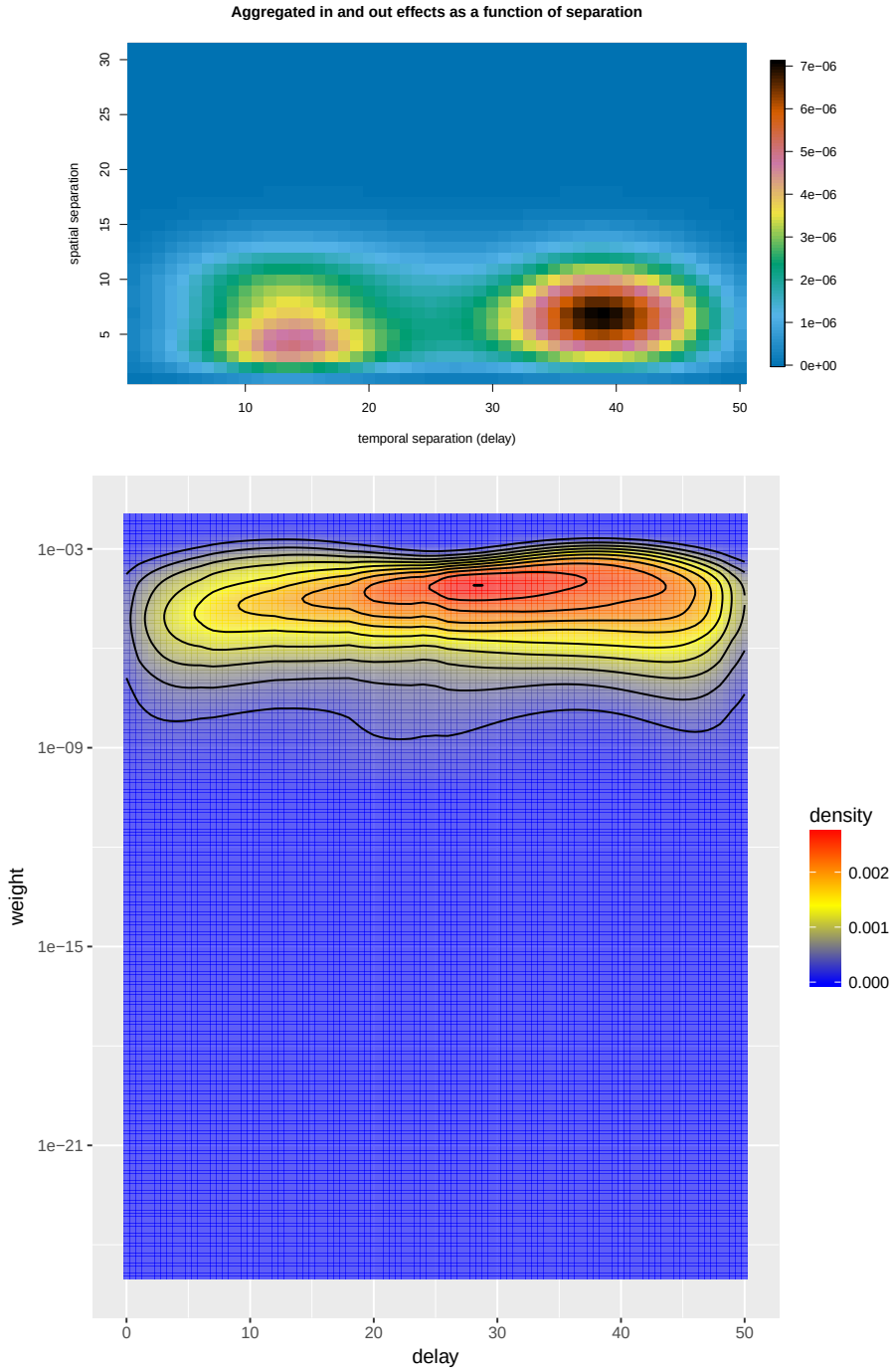

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
weight
density
delay

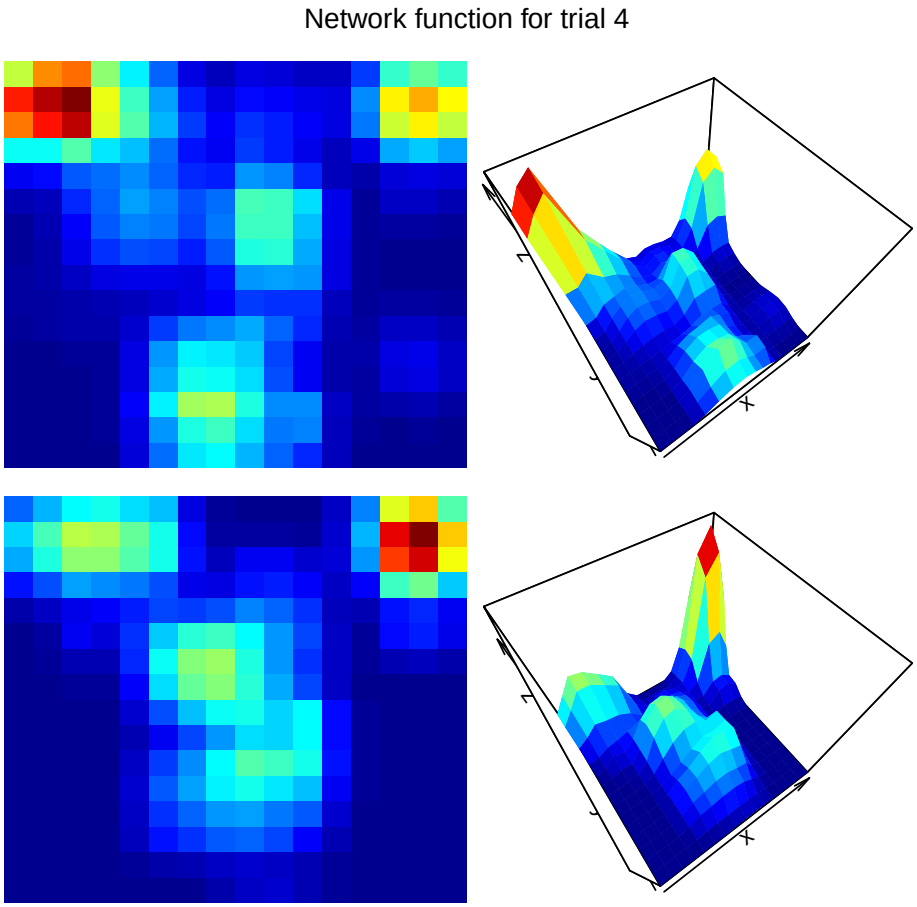
Network function for trial 2

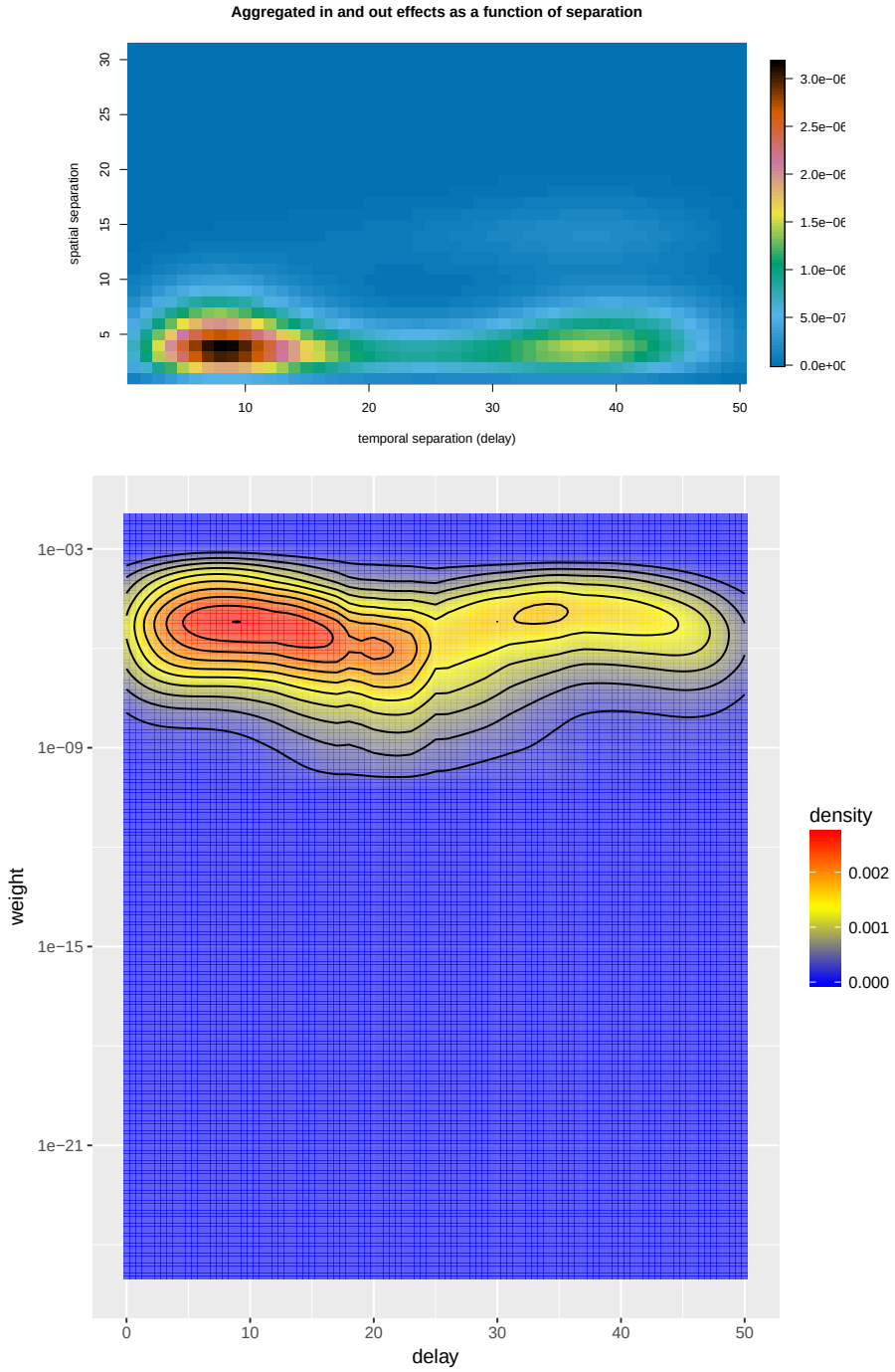


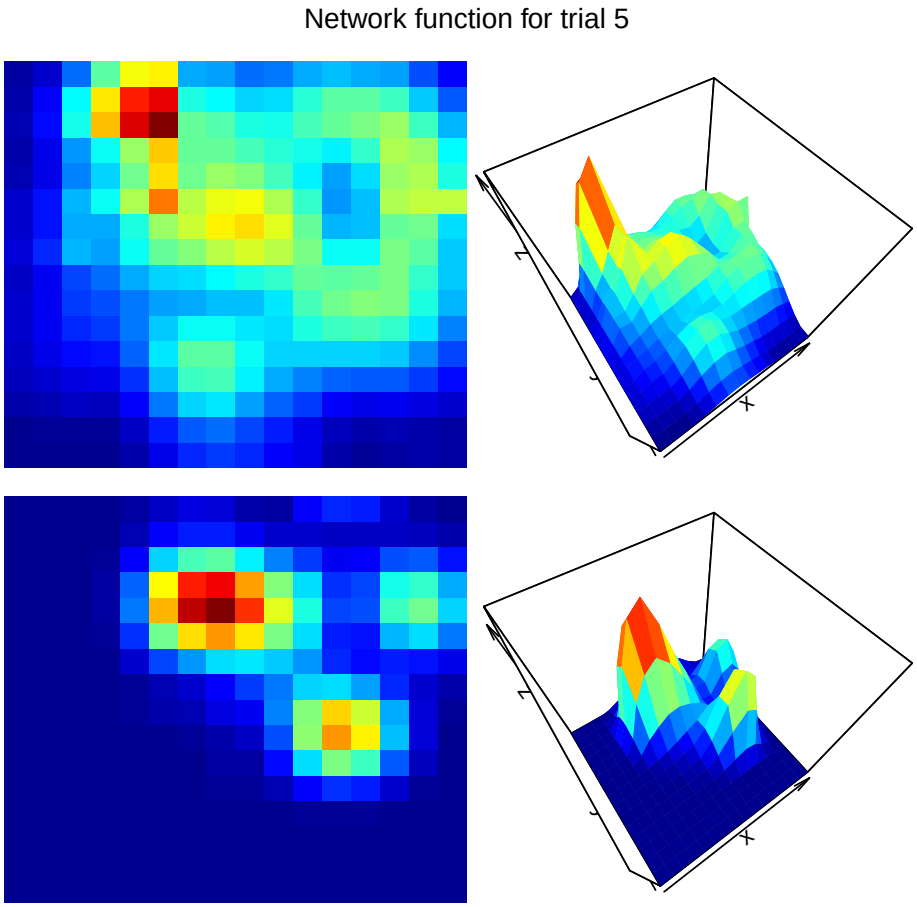

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
5e−06
4e−06
3e−06
2e−06
1e−06
0e+00
weight
delay
density
0.003
0.002
0.001
0.000

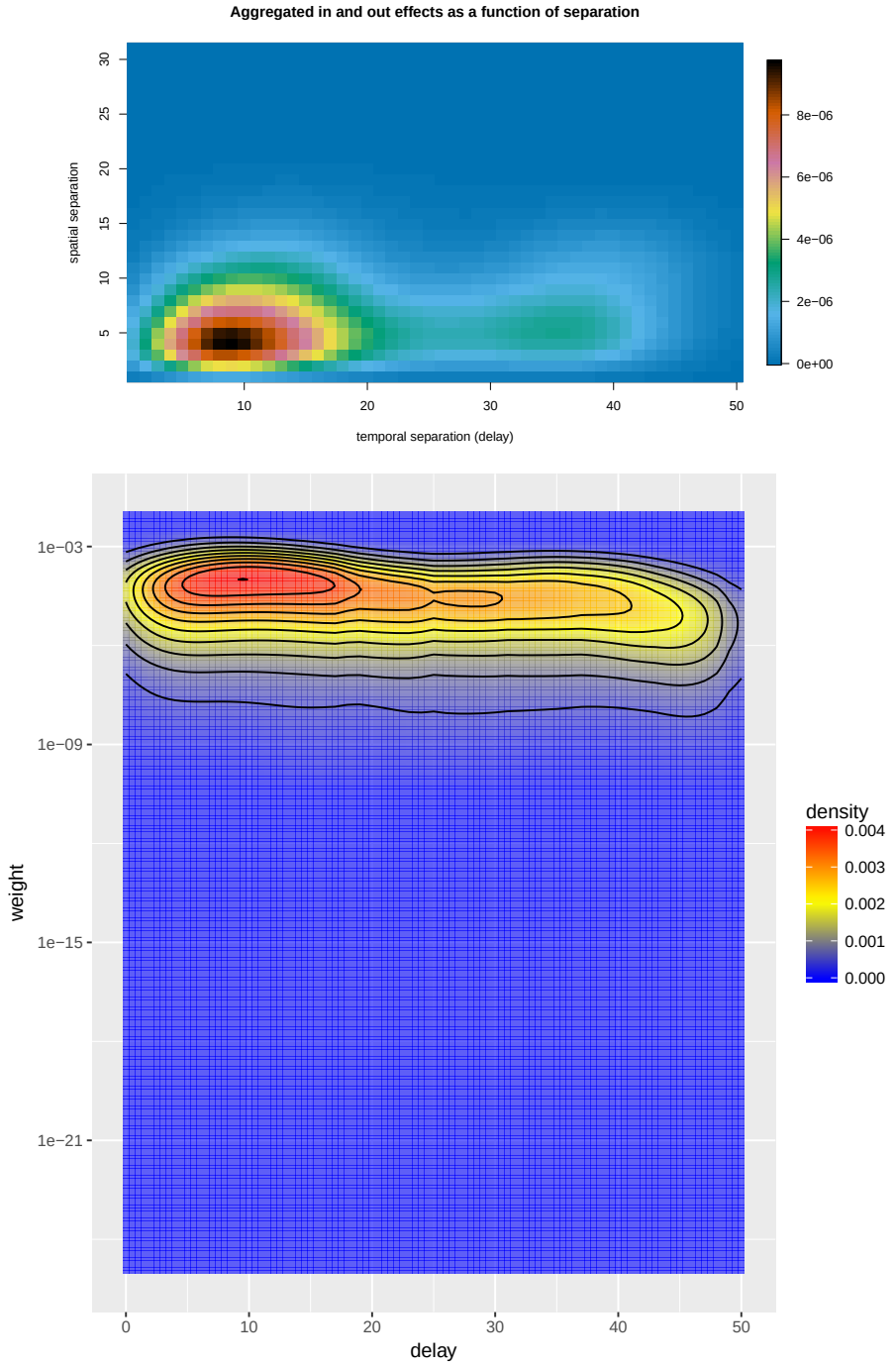

Network function for trial 3
Z
Y
X
Z
Y
X

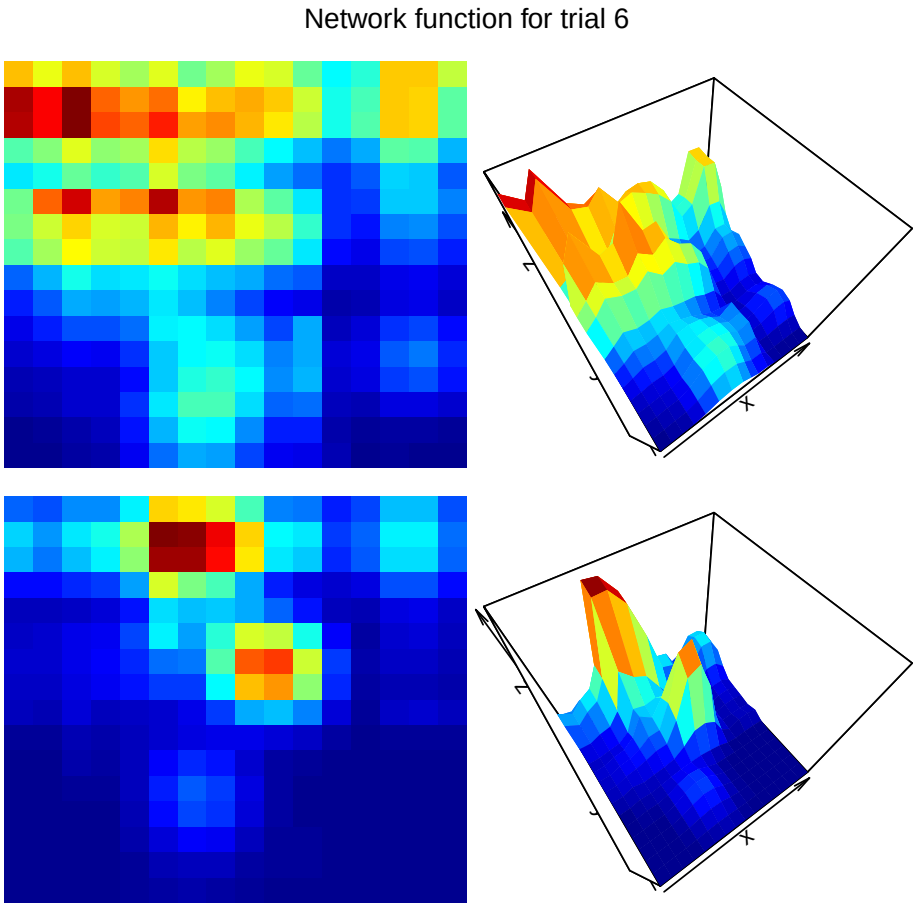

Aggregated in and out effects as a function of separation
spatial separation
5
10
15
20
25
30
temporal separation (delay)
10
20
30
40
50
7e−06
6e−06
5e−06
4e−06
3e−06
2e−06
1e−06
0e+00
weight
1e−03
1e−09
1e−15
1e−21
delay
0
10
20
30
40
50
density
0.002
0.001
0.000

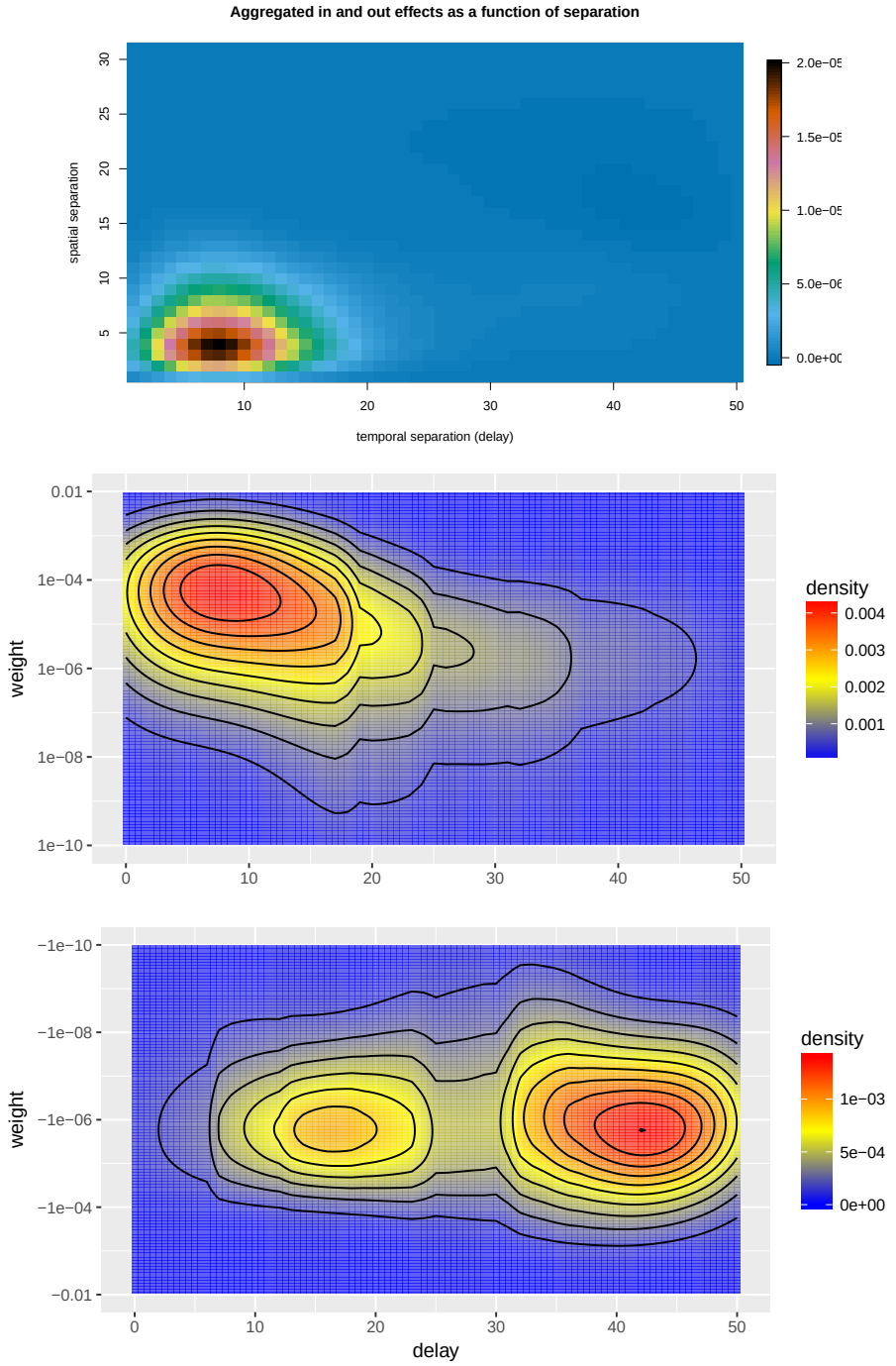

Network function for trial 4
Z
Y
X
Z
Y
X

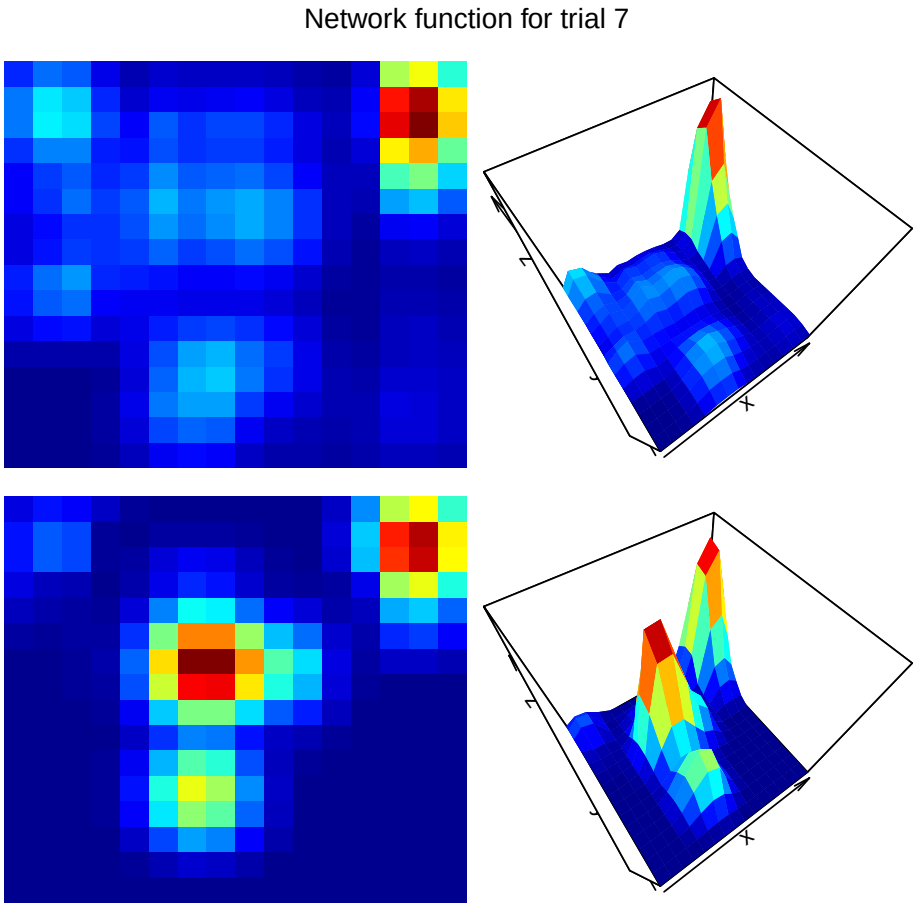

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
weight
delay
density
0.002
0.001
0.000

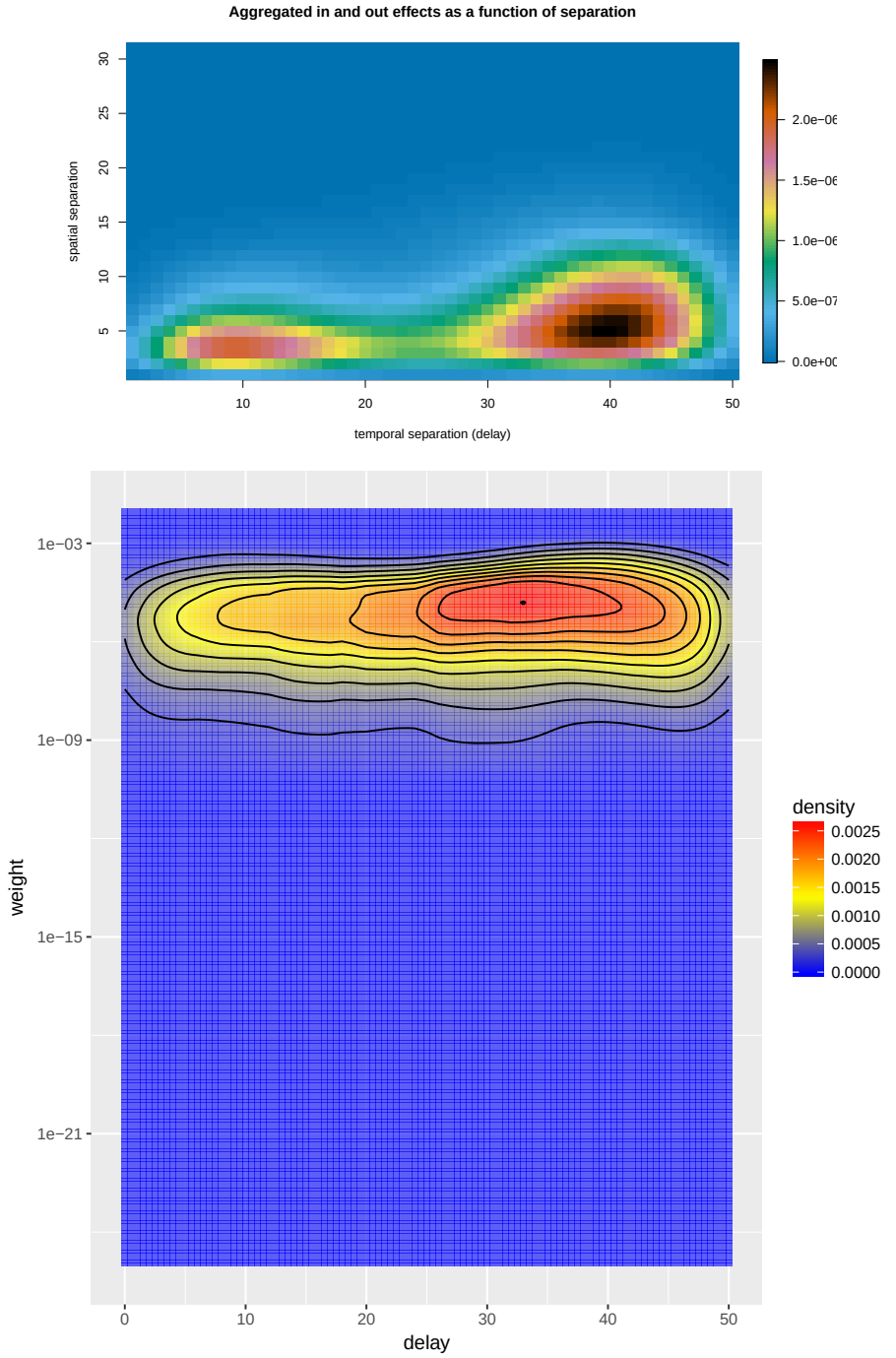

Network function for trial 5
Z
Y
X
Z
Y
X

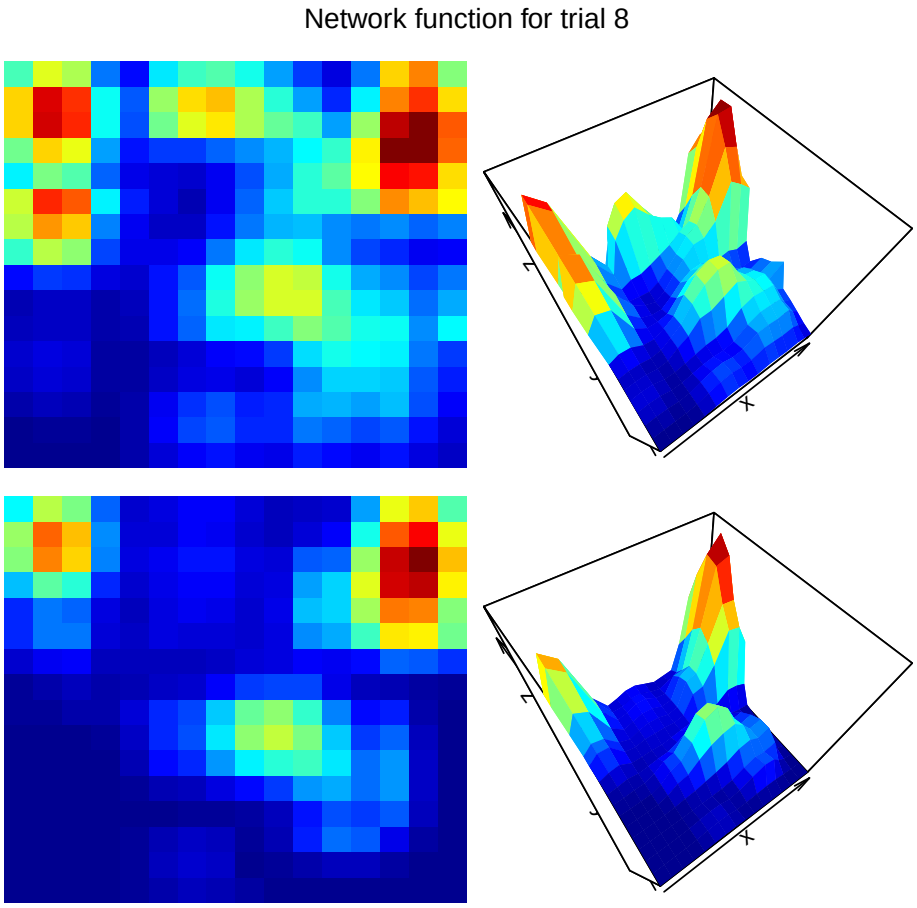

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
8e−06
6e−06
4e−06
2e−06
0e+00
weight
delay
density
0.004
0.003
0.002
0.001
0.000

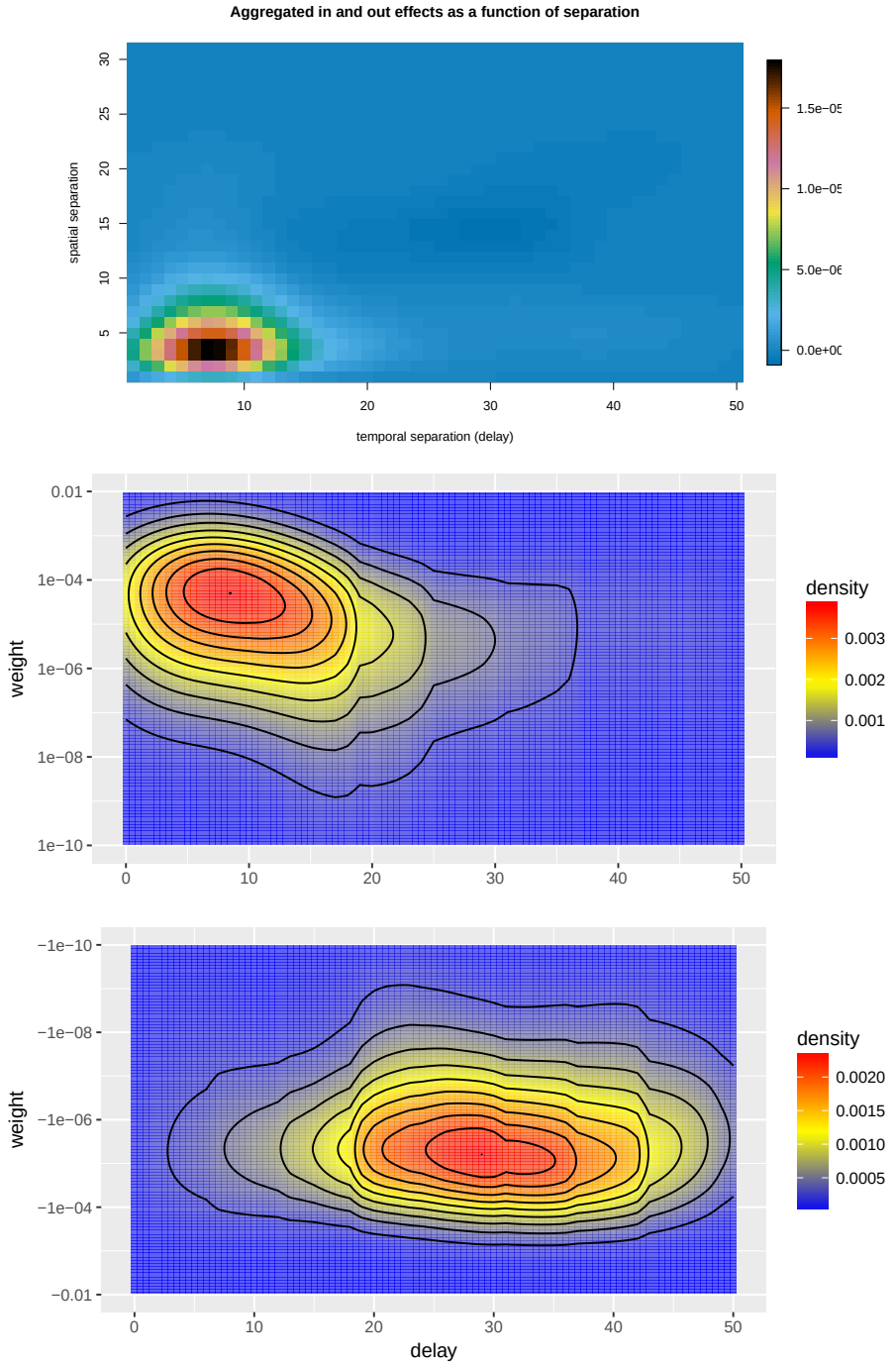

Network function for trial 6
Z
Y
X
Z
Y
X

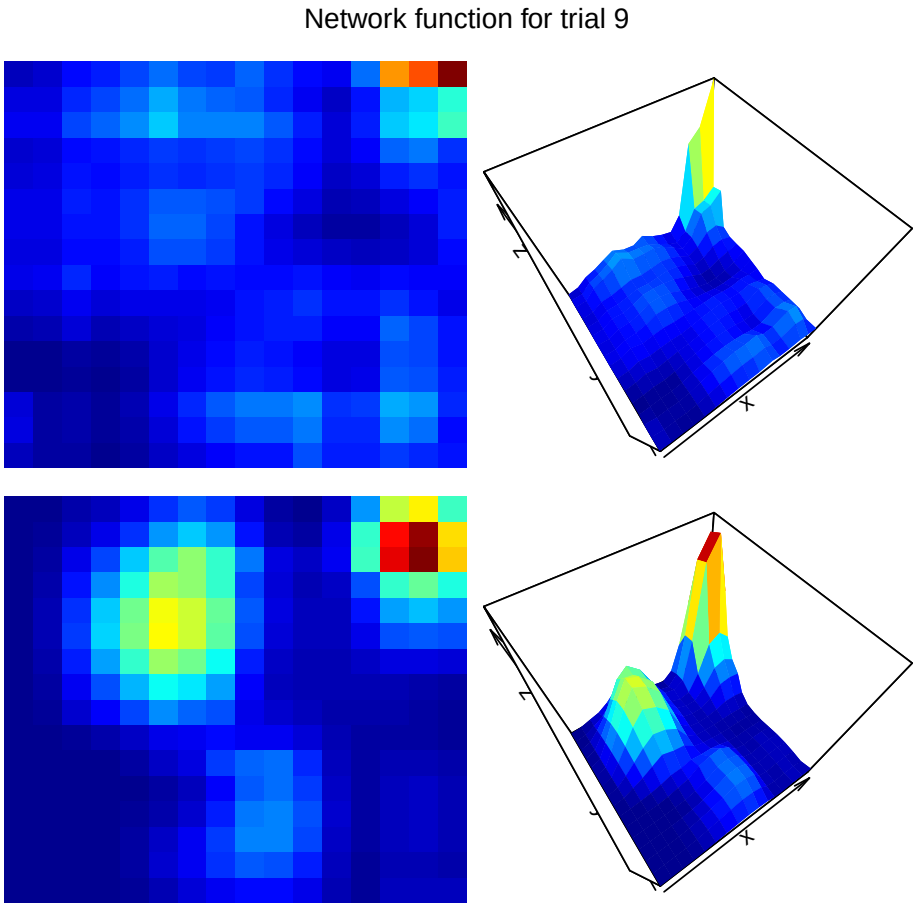

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
density
weight
delay

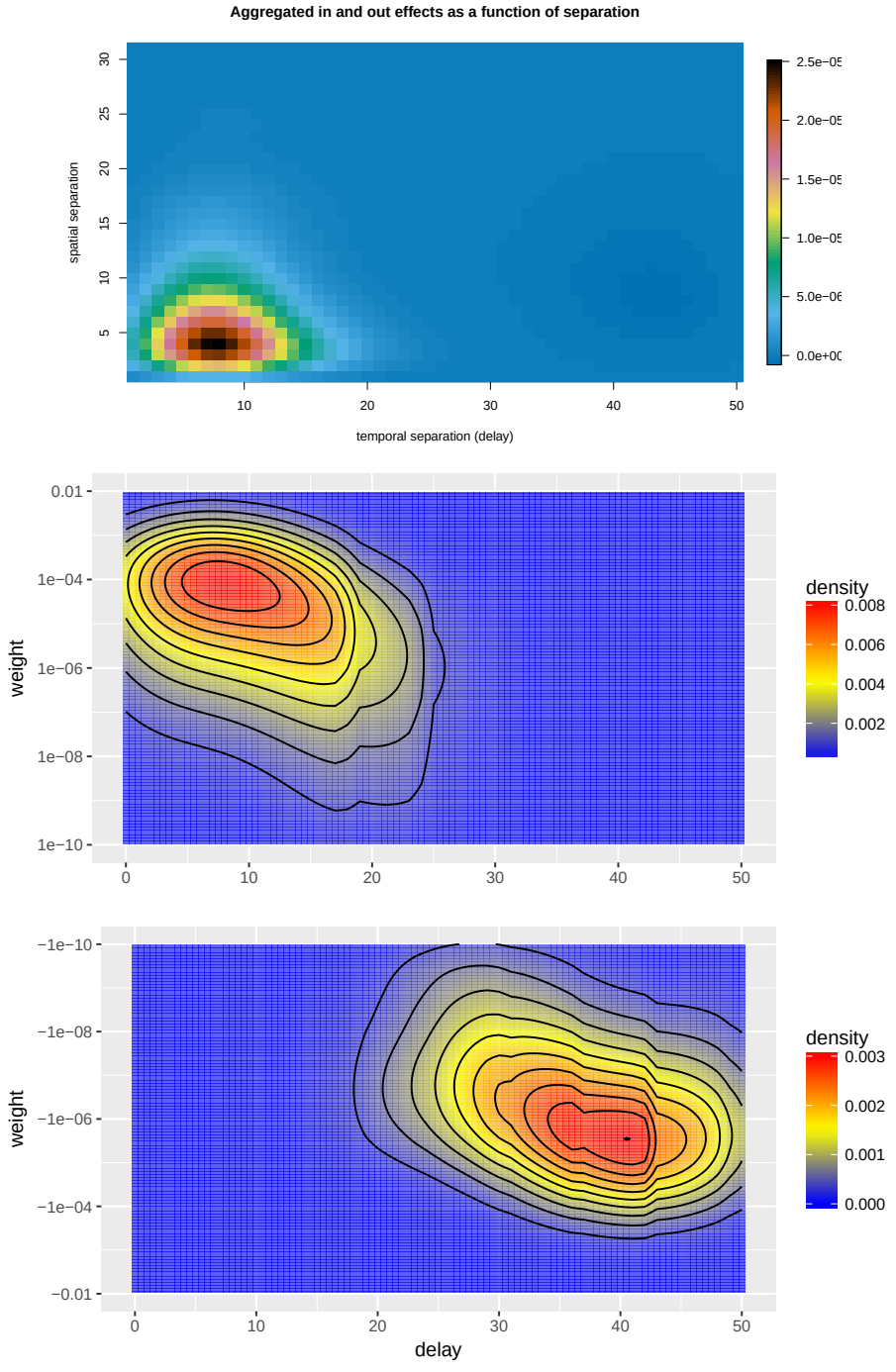

Network function for trial 7
Z
Y
X
Z
Y
X

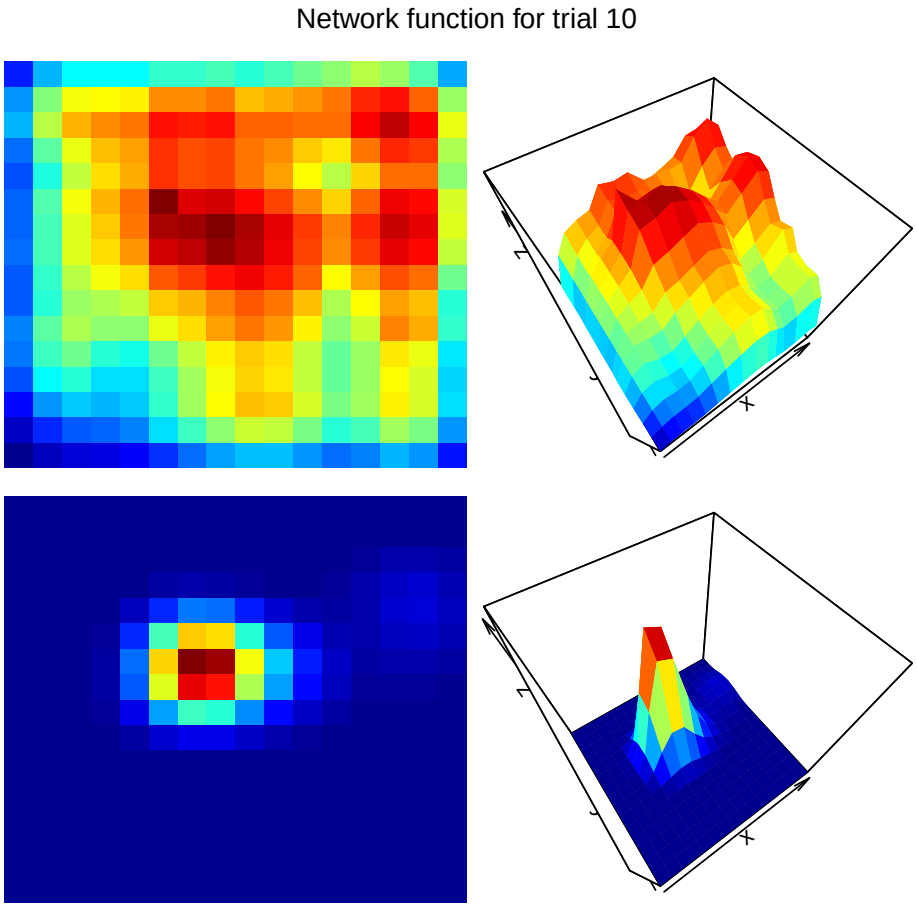

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
weight
delay
density
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000

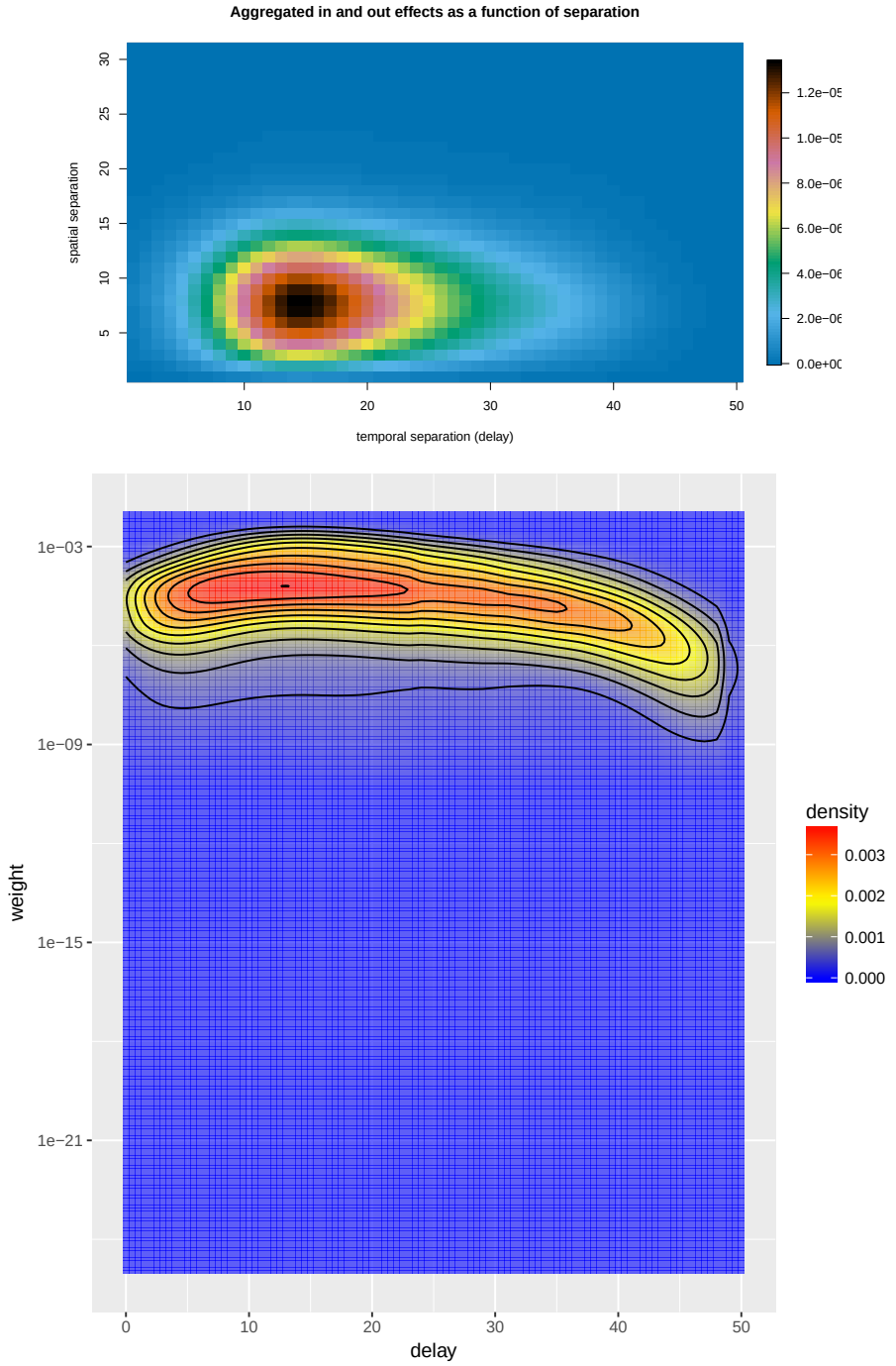

Network function for trial 8
Z
Y
X
Z
Y
X

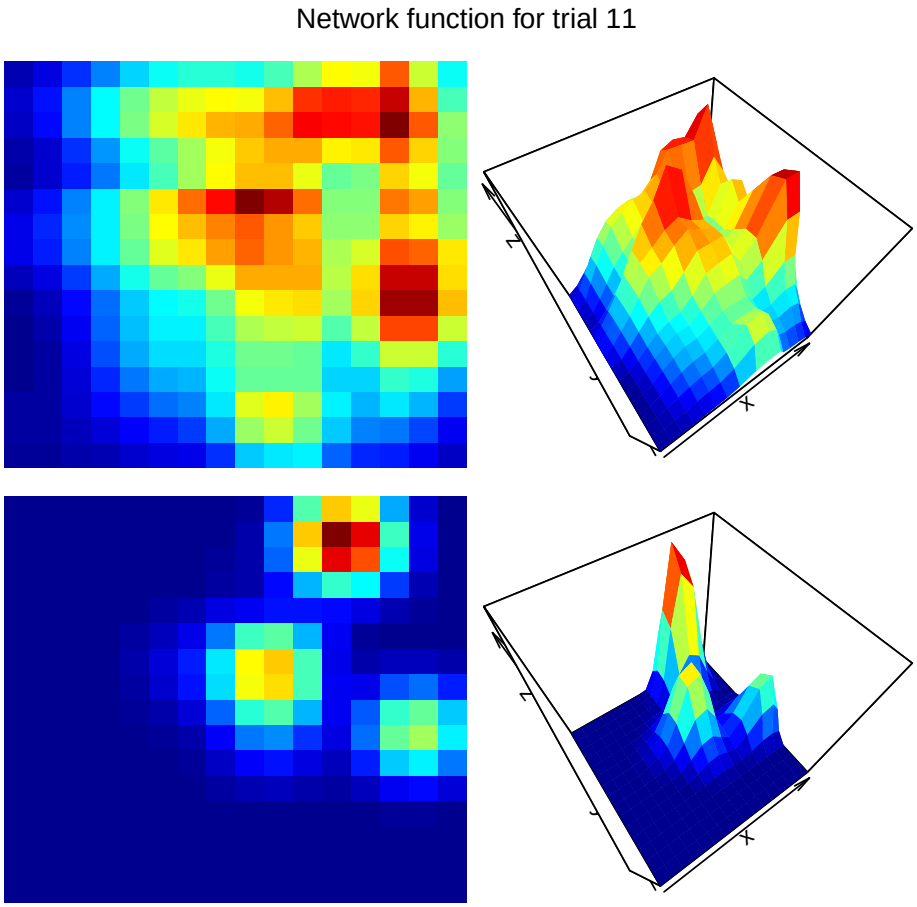

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
weight
density
delay

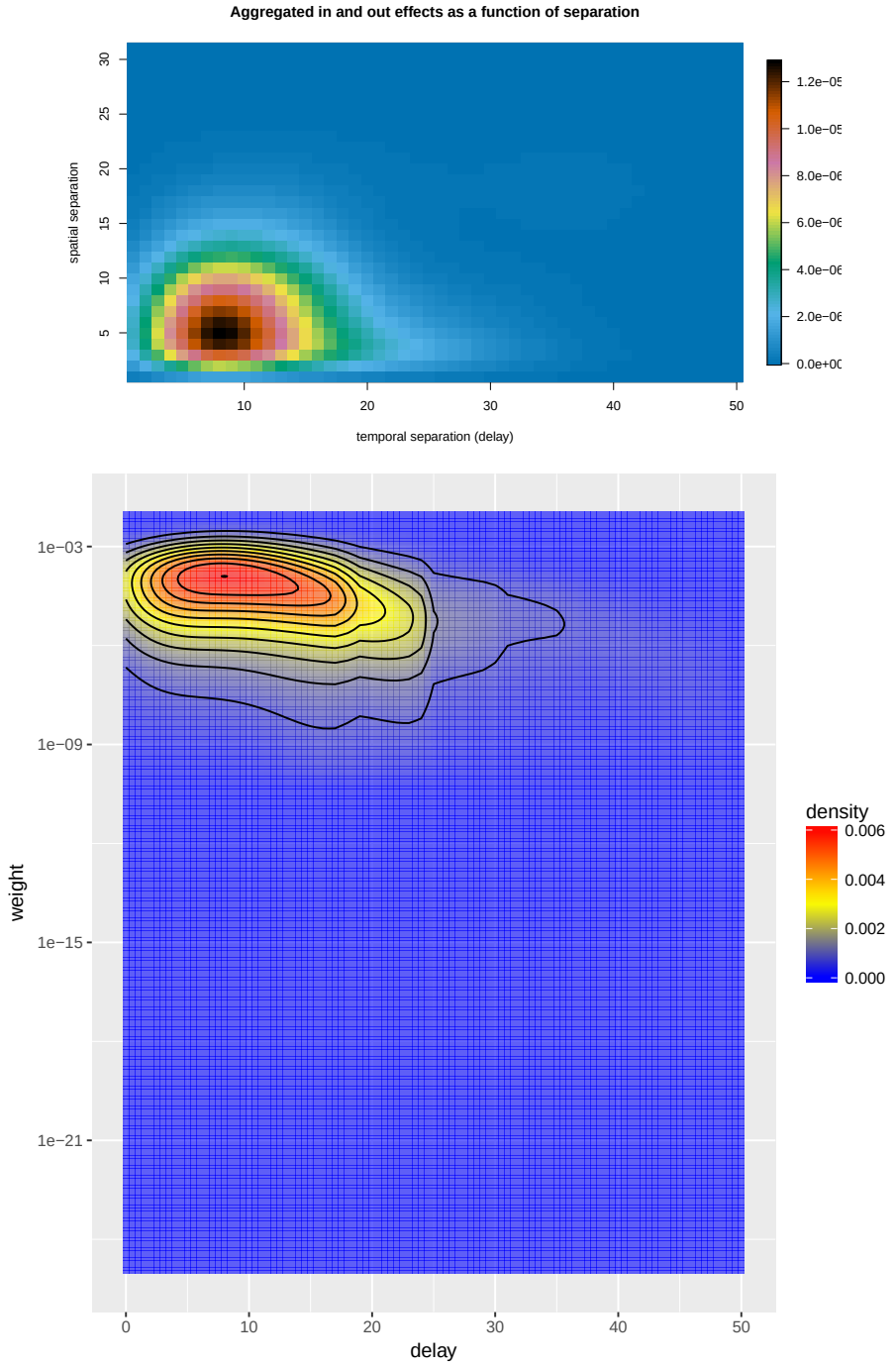

Network function for trial 9
Z
Y
X
Z
Y
X

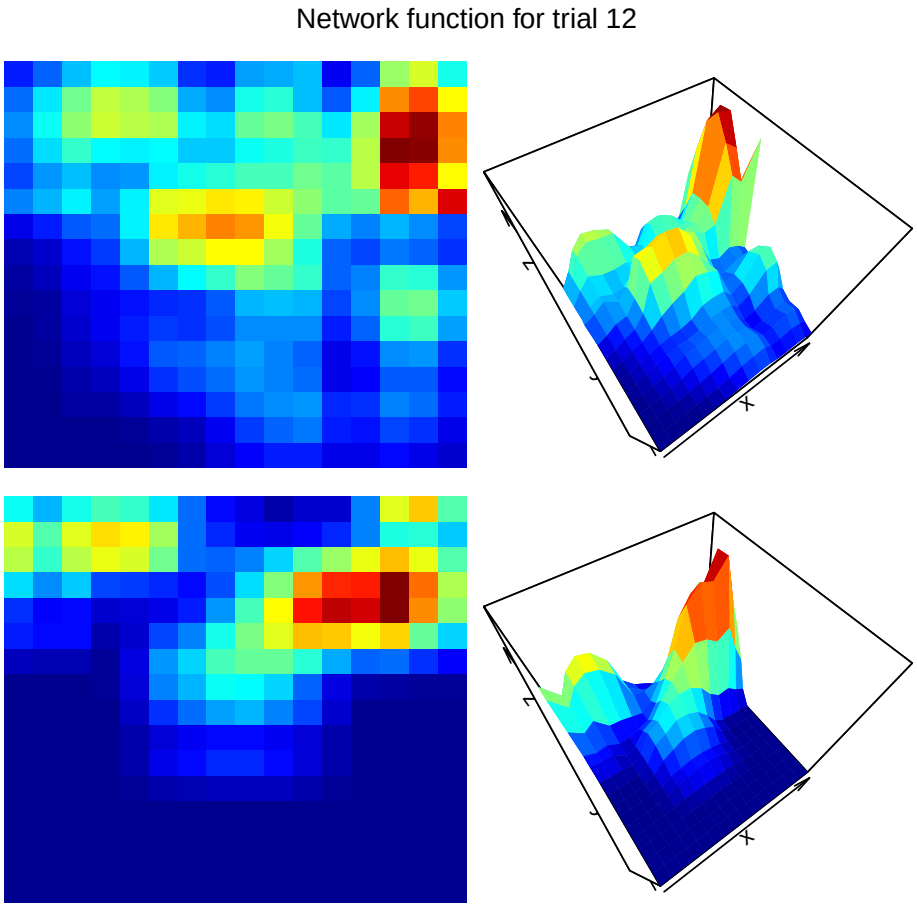

Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
weight
density
delay

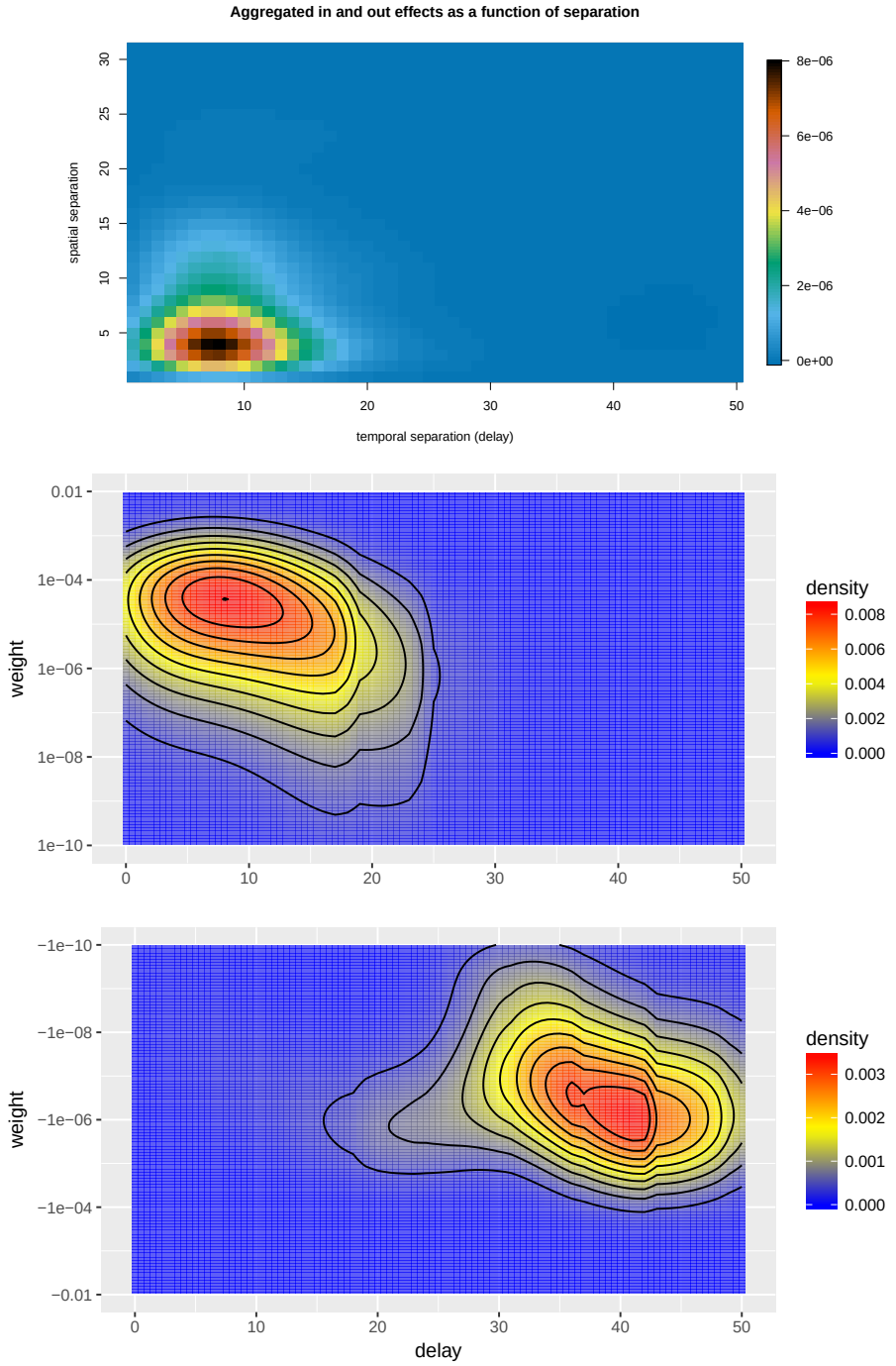

Network function for trial 10
Z
Y
X
Z
Y
X


Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
weight
delay
density
0.003
0.002
0.001
0.000


Network function for trial 11
Z
Y
X
Z
Y
X


Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
weight
delay
density
1e−03
1e−09
1e−15
1e−21
0.006
0.004
0.002
0.000


Network function for trial 12
Z
Y
X
Z
Y
X


Aggregated in and out effects as a function of separation
spatial separation
temporal separation (delay)
8e−06
6e−06
4e−06
2e−06
0e+00
weight
density
0.008
0.006
0.004
0.002
0.000
delay
0.003
0.001

REFERENCES

[1] Adler, R. J. and Taylor, J. E. (2009). *Random fields and geometry*. Springer Science & Business Media.

[2] beim Graben, P. and Potthast, R. (2009). Inverse problems in dynamic cognitive modeling. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **19** 015103.

[3] Bressloff, P. C. (2012). Spatiotemporal dynamics of continuum neural fields. *Journal of Physics A: Mathematical and Theoretical* **45** 033001.

[4] Bressloff, P. C. and Webber, M. A. (2012). Front propagation in stochastic neural fields. *SIAM Journal on Applied Dynamical Systems* **11** 708–740.

[5] Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of computational neuroscience* **8** 183–208.

[6] Buckwar, E. and Shardlow, T. (2005). Weak approximation of stochastic differential delay equations. *IMA journal of numerical analysis* **25** 57–86.

[7] 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.

[8] Chemla, S. and Chavane, F. (2010). Voltage-sensitive dye imaging: technique review and models. *Journal of Physiology-Paris* **104** 40–50.

[9] Coombes, S., beim Graben, P., Potthast, R. and Wright, J., eds. (2014). *Neural fields*. Springer, Heidelberg.

[10] Cox, S. G. (2012). Stochastic Differential Equations in Banach Spaces: Decoupling, Delay Equations, and Approximations in Space and Time. *PhD thesis*.

[11] 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 (Statistical Methodology)* **68** 259–280.

[12] Da Prato, G. and Zabczyk, J. (2014). *Stochastic equations in infinite dimensions*. Cambridge university press.

[13] Davis, R. A., Zang, P. and Zheng, T. (2016). Sparse Vector Autoregressive Modeling. *Journal of Computational and Graphical Statistics* **25** 1077-1096.

[14] De Boor, C. (1979). Efficient computer manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS)* **5** 173–182.

[15] Fan, J., Lv, J. and Qi, L. (2011). Sparse high dimensional models in economics. *Annual review of economics* **3** 291.

[16] Faugeras, O. and Inglis, J. (2015). Stochastic neural field equations: a rigorous footing. *Journal of Mathematical Biology* **71** 259–300.

[17] Friedman, J., Hastie, T. and Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* **9** 432–441.

[18] Harvey, M. A., Valentiniene, S. and Roland, P. E. (2009). Cortical membrane potential dynamics and laminar firing during object motion. *Frontiers in systems neuroscience* **3** 7.

[19] Lund, A. (2016). glamlasso: Penalization in Large Scale Generalized Linear Array Models R package version 2.0.1.

[20] Lund, A., Vincent, M. and Hansen, N. R. (0). Penalized estimation in large-scale generalized linear array models. *Journal of Computational and Graphical Statistics* **0** 0-0.

[21] Mao, X. (2007). *Stochastic differential equations and applications*. Elsevier.

[22] Markounikau, V., Igel, C., Grinvald, A. and Jancke, D. (2010). A Dynamic Neural Field Model of Mesoscopic Cortical Activity Captured with Voltage-Sensitive Dye Imaging. *PLoS Comput Biol* **6**.

[23] Peszat, S. and Zabczyk, J. (1997). Stochastic evolution equations with a spatially

homogeneous Wiener process. *Stochastic Processes and their Applications* **72** 187–204.

[24] Pinotsis, D. A., Moran, R. J. and Friston, K. J. (2012). Dynamic causal modeling with neural fields. *NeuroImage* **59** 1261 - 1274.

[25] Potthast, R. and beim Graben, P. (2009). Inverse Problems in Neural Field Theory. *SIAM Journal on Applied Dynamical Systems* **8** 1405-1433.

[26] 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.

[27] Rothman, A. J., Levina, E. and Zhu, J. (2010). Sparse multivariate regression with covariance estimation. *Journal of Computational and Graphical Statistics* **19** 947–962.

[28] Roxin, A. and Montbri, E. (2011). How effective delays shape oscillatory dynamics in neuronal networks. *Physica D: Nonlinear Phenomena* **240** 323 - 345.

[29] Sporns, O., Chialvo, D. R., Kaiser, M. and Hilgetag, C. C. (2004). Organization, development and function of complex brain networks. *Trends in cognitive sciences* **8** 418–425.

[30] Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* **58** 267-288.

[31] Touboul, J. (2014). Propagation of chaos in neural fields. *The Annals of Applied Probability* **24** 1298–1328.

[32] Valdés-Sosa, P. A., Sánchez-Bornot, J. M., Lage-Castellanos, A., Vega-Hernández, M., Bosch-Bayard, J., Melie-García, L. and Canales-Rodríguez, E. (2005). Estimating brain functional connectivity with sparse multivariate autoregression. *Philosophical Transactions of the Royal Society of London B: Biological Sciences* **360** 969–981.

University of Copenhagen,
Department of Mathematical Sciences,
Universitetsparken 5,
2100 Copenhagen Ø, Denmark
E-mail: adam.lund@math.ku.dk
niels.r.hansen@math.ku.dk

Chapter 7

Estimating Soft Maximin Effects in Heterogeneous Large-scale Array Data

Lund, A., S. W. Mogensen, and N. R. Hansen (2017). Estimating soft maximin effects in large-scale data. In preparation.

Estimating Soft Maximin Effects in Heterogeneous Large-scale Array Data

Adam Lund[1], Søren Wengel Mogensen and Niels Richard Hansen

University of Copenhagen, Department of Mathematical Sciences, Universitetsparken 5, 2100 Copenhagen Ø, Denmark

Abstract

In this paper we propose a new estimation problem - the soft maximin problem - along with a solution algorithm, aimed at extracting a common effect in large-scale heterogeneous data while being computationally attractive. The loss function in this problem - the soft maximin loss - relies on a smooth version of the max function leading to a smooth version of the (hard) maximin estimation problem from [1]. By showing that the soft maximin loss is strongly convex, we obtain a convergence result for a proximal gradient based solution algorithm for non-Lipschitz optimization problems. This algorithm can exploit the tensor array structure of our model leading to a computationally efficient design matrix free solution algorithm, implemented in the R package `SMMA`. We use this software to estimate a generic signal in a large-scale neuronal brain image data set and also demonstrate the approach on simulated data.

Keywords: sparse estimation, non-differentiable optimization, tensor-array structure, proximal gradient

1. Introduction

We consider the problem of extracting a common or generic signal from large scale heterogeneous array data organized in known groups. Especially, for a large-scale spatio-temporal array data set consisting of images of neuronal activity in 13 ferret brains recorded over time, the objective is to estimate a type of generic "brain signal" resulting from a visual stimulus. This

[1]Corresponding author. E-mail address: adam.lund@math.ku.dk (A. Lund).

data set contains several hundred thousand images recorded over multiple trials for each animal making the data both large-scale and heterogeneous.

We model this generic signal as the mean surface in a linear regression model using a tensor basis expansion to represent the smooth signal. Given the nature of the brain and the experimental setup the signal is presumably space and time localized meaning that it is zero much of the time over a large area of the visual cortex. Within our model setup this localization in turn translates to sparsity in the regression coefficient vector that we wish to estimate.

With $n_i \in \mathbb{N}$, $i = 1, 2, 3$, each trial in the data set consists of n_3 $n_1 \times n_2$ images and a total of $G \in \mathbb{N}$ trials are observed yielding $n_1 n_2 n_3 G$ observations. The inhomogeneities in the data stem from several sources. Temporal inhomogeneities can arise both across trials and animals owing to slightly varying experimental conditions as well as physical exertion. This potentially creates signal artifacts that are unrelated to the generic response signal of interest and can also change the shape of the latter signal. Furthermore spatial inhomogeneities are expected across animals corresponding to differences in the cytoarchitecture.

Being both large-scale and heterogeneous this data set fits well within the framework from [1]. Here a maximin method is proposed aimed at estimating a common effect in a large scale heterogeneous data set. Especially the maximin estimator extracts effects in the data that have the same sign across observations while setting effects that take opposite sign across observations to zero. Applied to our setup this would correspond to removing trial specific signals and instead extracting significant non-zero signals present across all trials, constituting the generic response signal in the visual cortex..

However, from a computational perspective the maximin method is challenging as it leads to a non-differentiable and non-separable convex optimization problem. Furthermore, this challenge is only exacerbated by applying the method in a large-scale context where memory requirements can make the problem computationally infeasible. Especially computer memory constraints can make it infeasible to store the design matrix in the working memory (the RAM). Furthermore operations involving the entire design matrix will be computationally expensive or perhaps even prohibitive.

Here we instead propose a related estimation problem that may be viewed as smooth approximation to the (hard) maximin problem from [1]. By using a type of soft max function we obtain an estimation problem - the soft maximin problem - that has a differentiable loss, the soft maximin loss. This in turn

leads to a related optimization problem that can be solved using efficient algorithms. Due to the large scale of our data we need an algorithm that can exploit the special tensor structure underlying our setup. As shown in [2] the proximal gradient algorithm is very well suited to exploit this kind of structure. However, while the soft maximin loss is very smooth its gradient is not Lipschitz continuous. This implies that we can not use a standard proximal gradient algorithm. Instead by establishing that the soft maximin loss is strongly convex we show that a non-monotone proximal algorithm proposed in [3] can solve the proposed soft maximin problem. By exploiting the tensor structure resulting form our data-model combination we obtain a computationally efficient design matrix free algorithm for solving the soft maximin problem in this setting. This algorithm is implemented as the R-package `SMMA` available from CRAN, see [4].

Using this software, we are able to extract one generic smooth signal from the brain image data set. This signal exhibits space and time localization with a clear depolarization occurring in an area corresponding to the center of field of view across all animals. Furthermore we test the approach on simulated data and present an example where the maximin estimator succeeds in extracting a common signal from noisy heterogeneous data while straight forward approaches like the cross group average or median fails.

2. Soft maximin effects for known groups

Following [1] we consider a mixed model for n response variables $Y_1, \ldots, Y_n$, given as

$$Y_i = X_i^\top B_i + \varepsilon_i. \tag{1}$$

Here X_i and $B_i \sim F_B$ are independent p-dimensional random vectors, F_B denotes the distribution of B_i, and ε_i is a zero-mean real random variable. The predictor variables $X_1, \ldots, X_n$ are independent with Gram population matrix Σ assumed to have full rank and we assume that ε_i and X_i are uncorrelated.

In this paper, we are interested in a setting with a known group structure. That is a known partition of the set of observations such that the random coefficient vector is the same for all observations within each partition element (group). More formally, there exists a known labelling function $\mathcal{G} : \{1, \ldots, n\} \to \{1, \ldots, G\}$, where $G \in \mathbb{N}$ is the cardinality of the partition, i.e. the number of groups. Then for $g \in \{1, \ldots, G\}$ let n_g be the number of observations in group g such that $\sum_g n_g = n$ and let $\mathcal{G}^{-1}(g) = \{i_1, \ldots, i_{n_g}\}$

denote the observation indices belonging to group g. Then for group g, $Y_g := (Y_{i_1}, \ldots, Y_{i_{n_g}})^\top$ denotes the group-specific $n_g \times 1$ response vector and by stacking the corresponding n_g group-specific predictors, $X_{i_1}^\top, \ldots, X_{i_{n_g}}^\top$ we obtain a group-specific $n_g \times p$ design matrix, $X_g := (X_{i_1} \mid \ldots \mid X_{i_{n_g}})^\top$. We then write the model for the gth group as

$$Y = X_g b_g + \epsilon, \tag{2}$$

indicating that $B_{i_1} = \cdots = B_{i_{n_g}} = b_g$, is the random coefficient for all n_g observations in group g. From [1], in the know groups setup, the empirical explained variance for group g using a $\beta \in \text{supp}(F_B)$ is then

$$\hat{V}_g(\beta) := \frac{1}{n_g}(2\beta^\top X_g^\top y_g - \beta^\top X_g^\top X_g \beta). \tag{3}$$

Now for $x \in \mathbb{R}^G$ and $\zeta > 0$ define the soft maximum function

$$s_\zeta(x) := \frac{\log(\sum_g e^{\zeta x_g})}{\zeta},$$

and let $s_{-\zeta}$ denote the corresponding soft minimum function. Letting $\hat{V}(\beta) := (\hat{V}_1(\beta), \ldots, \hat{V}_G(\beta))$ denote the vector of explained variances in the G groups we then propose maximizing the soft minimum of the explained variances as a function of β. Using that $s_{-\zeta}(x) = -s_\zeta(-x)$, this leads to the soft maximin estimator β_{smm} given by the problem

$$\beta_{smm} := \underset{\beta \in \mathbb{R}^p}{\arg\max}\, s_{-\zeta}(\hat{V}(\beta)) = \underset{\beta \in \mathbb{R}^p}{\arg\min}\, s_\zeta(-\hat{V}(\beta)). \tag{4}$$

In the following we shall refer to

$$l_\zeta(\beta) := s_\zeta(-\hat{V}(\beta))$$

as the soft maximin loss function.

We note that using l'Hôpistal's rule it follows that $s_\zeta(x) \to \max\{x\}$ and $s_{-\zeta}(x) \to \min\{x\}$ for $\zeta \to \infty$. Thus for $\zeta > 0$ the soft maximin problem (4) can be viewed as an approximation to the maximin problem proposed in [1]. We note, that where the only focus in the maximin problem is on the group

with the smallest explained variance, this group will be less dominating in the soft maximin problem (4). Especially, using that

$$\frac{\log(\frac{1}{G}\sum_g e^{\zeta x_g})}{\zeta} \to \frac{1}{G}\sum_g x_g$$

for $\zeta \to 0$, we see that $s_\zeta(x) \sim \frac{1}{G}\sum_g x_g + \frac{\log(G)}{\zeta}$ for small ζ. Thus the soft maximin can be seen as an interpolation between mean aggregation and max aggregation.

2.1. Smoothing model

For the neuronal brain data we are interested in modeling an underlying signal as a smooth function of time and space. For this data it is reasonable to use the individual trials to define a partition of the set of observations. We model the i_gth channel (or pixel) in the gth trial as

$$y_{i_g} = f_g(z_{i_g}) + \epsilon_{i_g}, \quad z_{i_g} \in \mathbb{R}^2 \times \mathbb{R}_+. \tag{5}$$

Here f_g is then a random trial specific smooth function modelling the brain response signal in the gth trial. We represent f_g using a basis expansion as

$$f_g(z) = \sum_{m=1}^{p} \Theta_{g,m}\phi_m(z), \tag{6}$$

for $\Theta_g = (\Theta_{g,1}, \ldots, \Theta_{g,p})$ a random vector, and $(\phi_i)_i$ a set of basis functions. Collecting the basis function evaluations into an $n_g \times p$ matrix $\Phi_g := (\phi_m(z_{i_g}))_{i_g,m}$, we can write the trial g model (5) as the linear regression (2) with $X_g := \Phi_g$ and $B_g := \Theta_g$. We note that for the brain image data model we have a deterministic design matrix X_g for all g. Furthermore as explained in section 4 below as we have the same channels and time points in each trial we may write $X_g = \Phi$ for all g.

Next we present a an example where this smoothing model have been fitted to real neuronal brain image data by solving a penalized version of the soft maximin problem (4). In subsection 4 below we present this general penalized estimation problem and establish a related solution algorithm.

3. Brian imaging data

The neuronal data was obtained using voltage-sensitive dye imaging (VSDI) and the experiment producing this data has previously been described in [5]. In short, parts of the visual cortex of a live ferret were exposed and stained with voltage-sensitive dye. By means of the dye, changes in neuronal membrane potential is translated into changes in the intensity of the fluorescent light. Recording the emitted fluorescent light using several hundred recording devices organized in a two-dimensional grid, produce an image of in vivo neuronal activity. Over the course of the experiment images are then recorded every 0.6136 ms producing a recording a film of neuronal activity. The experiment is divided into G trials. In each trial g two recordings are made; one where a visual stimulus is presented to the live ferret (a white square on a grey screen) and one where no stimulus is presented. These two recordings are then subtracted and normalized to give the final trial g recording. Thus for each trial g data is sampled in the 3-dimensional spatio-temporal grid yielding a 3-dimensional data array $\boldsymbol{Y}_g = (y_{i,j,k,g})_{i=1,j=1,k=1}^{n_1,n_2,n_3}$. Thus the entire data set can be organized in a 4-dimensional array $\boldsymbol{Y} = (\boldsymbol{Y}_g)_{g=1}^{G}$.

Given the experimental setup several sources of inhomogeneities are potentially present in the data. We list some here.

1. As the trials are recorded sequentially for each animal temporal inhomogeneities for each animal can arise resulting from a difference in response possibly due to fatigue.
2. Spatial inhomogeneities can arise due to differences in the cytoarchitectural borders between the animals causing misalignment problems.
3. The VSDI technique is very sensitive, see [6]. Consequently even small changes in the experiment surroundings could affect the recordings and create inhomogeneities in the data.
4. Physiological difference between animals and even in one animal over trials can also create inhomogeneities. Especially the heart beat affects the light emission by expanding the blood vessels in the brain. This creates a cyclic heart rate dependent artifact. A changing heart rate over trials for one animal as well as differences in heart rate between animals will cause inhomogeneous pulse artifacts in the data, see[5].
5. Ceteris paribus each animal could exhibit a slightly different response to the visual stimulus corresponding to an animal specific response signal.

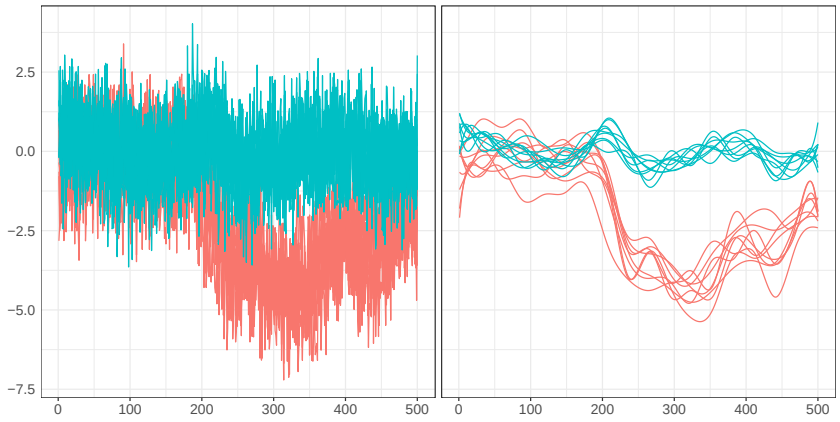


Figure 1: Evolution over time of light intensity measurements from a subset of channels from two different recordings (color). Left: raw data, right: smoothed data. One recording shows a clear depolarization following the stimulus whereas the other does not.

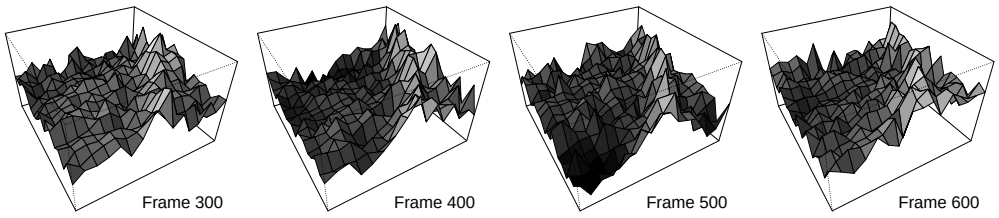


Figure 2: Four frames (time points) of a raw recording.

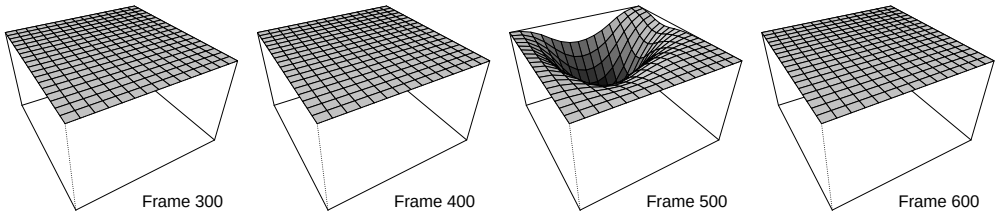


Figure 3: Four frames (time points) of a penalized soft maximin fit.

Given these considerations it seems reasonable to model the data in a framework that explicitly model potential inhomogeneities across trials as in

(5). For instance, following the stimulus, we would expect to see a reaction (a depolarization) in the visual cortex. However this is not is the case with all trials in the data set as can be gleaned from Figure 1. Figure 1 shows the individual analysis of two trial recordings obtained by fitting each trial to the model (5) along with the raw recordings. While one recording (red) shows clear evidence of depolarization the other recording does not. This seems to indicate that systematic noise components, i.e. artifacts as in 4 above, are present in the data as white noise would hardly mask the depolarization entirely.

These heterogeneity issues lead the authors who originally recorded this data to only consider a subset of the recordings with a significantly increased depolarization in the visual cortex following the stimulus, see [5]. Here however, we will fit the entire data set using the maximin estimator.

4. Solving the penalized soft maximin problem

Next we first present an algorithm for solving a penalized version of the soft maximin problem. Then we formulate the smoothing model from subsection 2.1 as a linear model with array-tensor structure and discuss how this structure might be exploited in connection with the proposed solution algorithm. This in turn leads to an estimation procedure for array-tensor structured models it scales well with the size of data as it completely avoids the memory issues plaguing existing methods when applied to problems with this structure.

4.1. Penalized soft maximin problem

To obtain space a time localization of the signal f_g from (5) we may use basis functions with compact support and then use a sparsity inducing penalty to remove basis functions in area with little signal. This idea leads us to consider a regularized version of the estimation problem (4) given by

$$\min_{\beta \in \mathbb{R}^p} l_\zeta(\beta) + \lambda J(\beta), \tag{7}$$

where J is a proper convex function. Choosing lasso penalty i.e. $\|\cdot\|_1$ (see [7]), solving (7) will result in a sparse estimates of β which in the smoothing model from section 2.1 translates into a space and time localized generic signal across all trials. In the following we let $F_\zeta := l_\zeta + \lambda J$ denote the soft maximin objective.

Compared to the penalized hard maximin problem in [1], the problem (7) is easier to solve. Especially we note that the loss function in the hard maximin problem is not only non-differentiable but also non-separable. This means that a computationally efficient algorithm like the coordinate descent algorithm cannot be used to solve the hard maximin problem, see [8]. We also note that in the setting from subsection 2.1, with fixed design and known groups, the solution path of the the hard maximin problem is piecewise linear in λ and thus may be solved using a method like LARS, see [9]. However, the LARS algorithm scales poorly with the size of data and in particular cannot exploit the structure of the data model combination we have in mind.

In contrast since the soft maximin loss is differentiable and convex we can use the coordinate descent algorithm to solve the problem (7) whenever J is separable. However solving (7) for a large scale data set still entails some challenges. Especially even though the coordinate descent algorithm has proven to be highly computationally efficient in solving problems like (7) (see e.g. [10]) it is not very well suited to exploit problems with a tensor structured design matrix. In this setting the coordinate descent algorithm is both very memory inefficient and also computationally inefficient as demonstrated in [2]. Next we propose proximal gradient algorithm the can always be used to solve (7) and in particular, for tensor structured design matrices, both highly computationally efficient as well as memory efficient.

4.2. Proximal gradient based algorithm

The problem (7) is easier to solve since the soft maximin loss l_ζ smooth especially it is C^∞. Furthermore it has the the following properties.

Proposition 1. *Let $h_g : \mathbb{R}^p \to \mathbb{R}$ be strongly convex for each $g \in \{1, \dots, G\}$ and twice differentiable . Then $h_\zeta : \mathbb{R}^p \to \mathbb{R}$ given by*

$$h_\zeta(\beta) := \frac{\log(\sum_{g=1}^{G} e^{\zeta h_g(\beta)})}{\zeta}$$

is strongly convex, $e^{\zeta h_\zeta}$ is strongly convex and especially the soft maximin loss l_ζ is strongly convex. Furthermore the gradient of h_ζ is given by

$$\nabla h_\zeta(\beta) = \frac{\sum_{g=1}^{G} e^{\zeta h_g(\beta)} \nabla h_g(\beta)}{e^{h_\zeta}}. \tag{8}$$

Proof. See Appendix A. □

An underlying assumption when applying the proximal algorithm is that the loss is Lipschitz continuously differentiable i.e. have a Lipschitz continuous gradient. From the gradient expression (8) it follows that ∇l_ζ is not Lipschitz continuous. This means that we cannot use a standard proximal algorithm to solve (7).

However using Proposition 1 and the fact that l_ζ is C^∞, hence locally Lipschitz continuously differentiable, we may use the non-monotone proximal algorithm from [3] presented next.

Algorithm 1 NPG

Require: β^0, $L_{max} \geq L_{min} > 0$, $\tau > 1$,$c > 0$, $M \geq 0$ arbitrarily

1: **for** $k = 0$ to $K \in \mathbb{N}$ **do**
2: choose $L_k \in [L_{min}, L_{max}]$
3: solve $\beta := p_{J/L_k}(\beta^{(k)} - \frac{1}{L_k}\nabla f(\beta^{(k)}))$
4: **if** $F_\zeta(\beta) \leq \max_{[k-M]_+ \geq i \geq k} F_\zeta(\beta^{(i)}) - c/2\|\beta - \beta^{(k)}\|$ **then**
5: $\beta^{(k+1)} := \beta$
6: **else**
7: $L_k := \tau L_k$ and go to 3.
8: **end if**
9: **end for**

Here step 3 is the proximal step with $p_{\delta J}$ denoting the proximal operator, see section Appendix C. We also note that for $M = 0$ the algorithm is in fact monotone.

Now by using Proposition 1 we can show the following convergence result for the algorithm when applied to the problem (7).

Proposition 2. *Let $(\beta^{(k)})_k$ be a sequence of iterates generated when applying Algorithm 1 to the soft maximin problem* (7)*. Then $\beta^{(k)} \to \beta^*$ where β^* is a critical point of F_ζ.*

Proof. See Appendix A □

Thus using Algorithm 1 we can solve the soft maximin problem (7) for any type of design matrix X_g in the setting with know group structure setting. However the main motivation behind using this algorithm is it that each iteration involves

1. evaluation of the gradient ∇l_ζ.

2. evaluation of the proximity operator p_{J/L_k}.
3. evaluation of the objective function F_ζ.

If we choose J as the ℓ_1-penalty then evaluating the proximity operator just entails soft thresholding on the parameter vector. Furthermore, for the penalized soft maximin problem (7), evaluating the gradient and the soft maximin objective essentially comes down to computing the design matrix vector product $X_g\beta$. This implies that if we can efficiently compute this product we can enhance the performance of the NPG algorithm. Especially the NPG algorithm is well suited to exploit the tensor structure of the smoothing model as discussed next.

4.3. Tensor array smoothing

Considering the discussion of the neuronal of data in section 3 we see that this data is sampled in a 3-dimensional grid that we can write as

$$\mathcal{X}_1 \times \mathcal{X}_2 \times \mathcal{X}_3 \tag{9}$$

where $\mathcal{X}_i = \{x_1, \ldots, x_{n_i}\}$ with $x_i < x_{i+1}$ yielding a corresponding 3-dimensional array $\boldsymbol{Y}_g$ for each trial g. Preserving this structure when formulating the mixed model setup for know groups for smoothing as in section 2 naturally leads to a tensor structured problem

For a smoothing model like (5) the tensor structure arise as a consequence of the specification of the basis functions. Especially the 3-variate basis functions appearing in (5) can be specified in terms of three (marginal) sets of univariate functions. Assuming $p := p_1p_2p_3$ it is possible, using the tensor product construction, to specify the mth basis function in (6) as

$$\phi_m := \phi_{1,m_1}\phi_{2,m_2}\phi_{3,m_3}, \tag{10}$$

where $\phi_{j,m} : \mathbb{R} \to \mathbb{R}$ for $j = 1, \ldots, 3$ and $m = 1, \ldots, p_j$ is a marginal basis function. With this specification it is then possible to organize the random basis coefficient in a corresponding $p_1 \times p_2 \times p_3$ array $\boldsymbol{\Theta}_g = (\Theta_{j_1,j_2,j_3})_{j_1=1,j_2=1,j_3=1}^{p_1,p_2,p_3}$

Then for each j evaluating each of the p_j univariate functions in the n_j points in $\mathcal{X}_j$ results in an $n_j \times p_j$ marginal design matrix $\Phi_j = (\phi_{j,m}(x_k))_{k,m}$.

By forming the tensor product (Kronecker product) of these marginal design matrices, we obtain the $n \times p$ tensor product matrix

$$\Phi := \Phi_3 \otimes \Phi_2 \otimes \Phi_1, \tag{11}$$

as the design matrix for the gth trial data. Thus this smoothing model has a (one component) array-tensor structure and as discussed in Appendix B, using the rotate H transform ρ from [11] it follows that we can write the trial g model (5) as a one component linear array model

$$\boldsymbol{Y}_g = \rho(\Phi_3, \rho(\Phi_2, \rho(\Phi_1, \boldsymbol{\Theta}_g))) + \boldsymbol{E}_g$$

where $\boldsymbol{E}_g$ is a $n_1 \times n_2 \times n_3$ array containing the noise terms.

For any linear model with this structure we can compute the design matrix vector product without having access to the design matrix. Thus by exploiting this structure we avoid memory issues when fitting large-scale data implying that we can fit much more data compared to a setting with no such structure. Furthermore, as an additional bonus, the computation of the design matrix-vector product is performed much more efficiently by the rotated H transform ρ. Following [2] we can therefore enhance the NPG algorithm when applied to the soft maximin problem (7) for array data with tensor structured design matrix resulting in a computationally efficient numerical procedure with a small memory foot print.

5. Simulation study

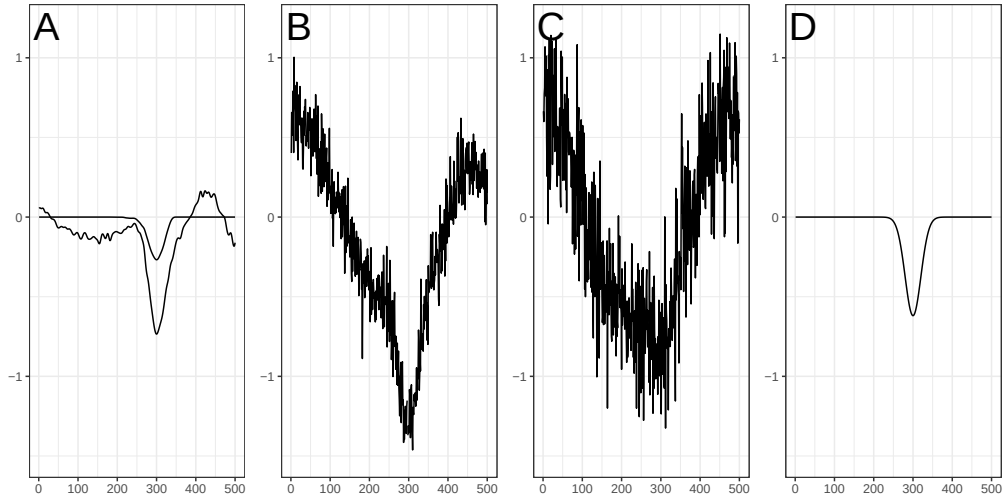


Figure 4: All four subplots are of a single channels evolution over time. A: maximin prediction for two values of the penalization parameter. B: mean prediction. C: median prediction. D: common component.

The purpose of the simulation study was to examine the maximin effects' ability to extract a true signal from noisy data with systematic noise components. We compared it to simple aggregation methods such as taking the entry-wise mean and median obtaining a mean or median recording from simulated data. Two main properties are appealing, both the ability to extract a common component when one is present and the ability to not exhibit any signal when none is present. The latter ability is somewhat implied by Theorem 1 in [1] as adding more observations will give new maximin effects that are (weakly) closer to the origin.

For the simulation study we generated data which is somewhat similar to the ferret data. Each recording consisted of a number of spatio-temporal fluctuations and a common component was added to each recording. Data was simulated under a true model, in the sense that each recording corresponded to noisy observations of data from some coefficient vector where the design matrix was known. The common component resembled a depolarization (see row B in Figure 6) and was temporally sparse with maximal signal at time point (frame) 300. A total of 50 recordings were simulated.

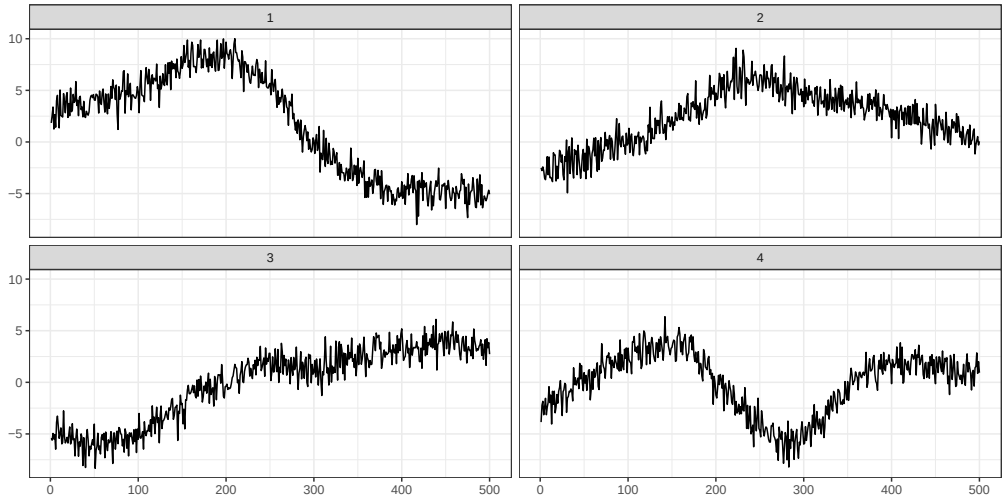


Figure 5: Raw data from a single channel as it evolves over time. Each subplot is a recording.

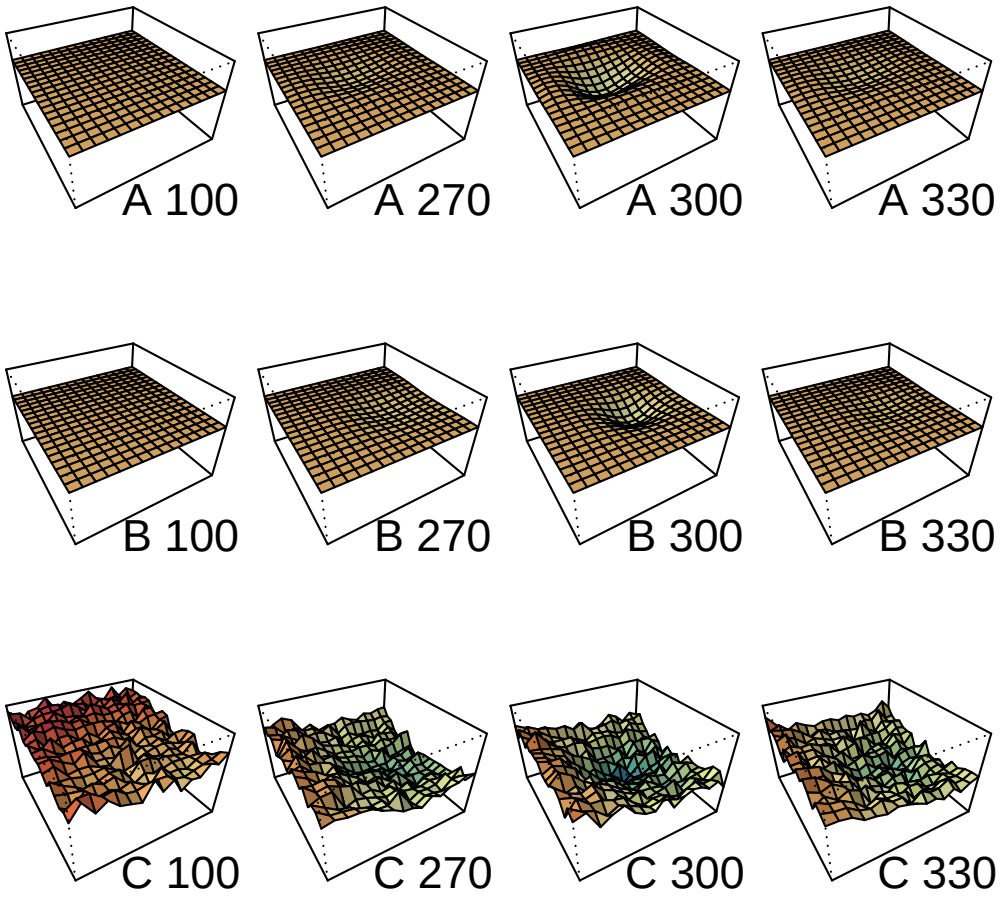


Figure 6: 3D visualization at four fixed time points (frames 100, 270, 300, and 330. A: common component. B: maximin prediction (penalized). C: mean prediction.

From Figures 4 and 5 it is clear that the scale of the raw data is considerably larger than the scale of the common component (the same channel is used for both figures). We can also see that the random fluctuations are so large that for some recordings the common component (the "dent" in Figure 4, D) is not visible in the raw data.

In Figure 4 predictions for a single channel is displayed over time. Due to the large and systematic noise components both the entry-wise mean and

median show fluctuations that are considerably larger than that of the common component. As expected, the maximin effects are more conservative (smaller deviations from zero) and by increasing the penalty parameter we can obtain a temporally sparse signal. Figure 6 is a 3D visualization of the same simulation example. It seems clear from both figures that e.g. the mean is showing the common signal but it is superimposed on other signals which means that without prior knowledge we would not be able to distinguish the common component from what is essentially systematic noise components that are not present in all recordings.

Appendix A. Proofs

Proof of Proposition 1. We note that for a twice continuously differentiable function f and $\nu > 0$ it holds that

$$f \text{ strongly convex} \Leftrightarrow \nabla^2 f(x) - \nu I \text{ positive definite} \Leftrightarrow f(x) - \|x\|_2^2 \text{ convex.}$$

Now let $\tilde{h}_g(\beta) := \zeta h_g(\beta) - \|\beta\|_2^2$ and consider the function

$$h(\beta) := \zeta h_\zeta(\beta) - \|\beta\|_2^2 = \log\left(\sum_{g=1}^{G} e^{\zeta h_g(\beta) - \|\beta\|_2^2}\right) = \log\left(\sum_{g=1}^{G} e^{\tilde{h}_g(\beta)}\right).$$

We note that $\tilde{h}_g$ is convex as h_g is strongly convex. Using this and Hölders inequality with the $\ell_{1/\alpha}$ and $\ell_{1/(1-\alpha)}$ norms for $\alpha \in (0,1)$, we get

$$\begin{aligned} e^{h(\alpha\beta+(1-\alpha)\beta')} &= \sum_{g=1}^{G} e^{\tilde{h}_g(\alpha\beta+(1-\alpha)\beta')} \\ &\leq \sum_{g=1}^{G} e^{\alpha\tilde{h}_g(\beta)+(1-\alpha)\tilde{h}_g(\beta')} \\ &\leq \left(\sum_{g=1}^{G} e^{\tilde{h}_g(\beta)}\right)^{\alpha} \left(\sum_{g=1}^{G} e^{\tilde{h}_g(\beta')}\right)^{1-\alpha}. \end{aligned}$$

Thus by taking the logarithm on both sides it follows that h is convex hence ζh_ζ is strongly convex implying h_ζ is strongly convex.

Now as h_ζ is strongly convex we have that $\nabla^2 h_\zeta(\beta) - \nu I$ is positive semi definite for some $\nu > 0$. With $\nabla^2 h_\zeta(\beta)$ the Hessian of h_ζ, the Hessian of $e^{h_\zeta(\beta)}$ is

$$\nabla^2 e^{h_\zeta(\beta)} = \nabla^2 h_\zeta(\beta) e^{h_\zeta(\beta)} + \nabla_\beta h_\zeta(\beta) \nabla_\beta h_\zeta(\beta)^\top e^{h_\zeta(\beta)}$$

where $\nabla_\beta h_\zeta(\beta)\nabla_\beta h_\zeta(\beta)^\top e^{h_\zeta(\beta)}$ is positive semidefinite for all β. As $m := \min_\beta h_\zeta(\beta) \in \mathbb{R}$, we have $e^{h_\zeta(\beta)} \geq e^m > 0$ for all β and must have $\nabla^2 e^{h_\zeta(\beta)} - \tilde{\nu} I$ is positive definite for some $\tilde{\nu} > 0$, showing that e^{h_ζ} is strongly convex.

Especially, we note that

$$\begin{aligned} -\hat{V}_g(\beta) &= -\frac{1}{n_g}(2\beta^\top X_g^\top Y_g - \beta^\top X_g^\top X_g \beta + Y_g^\top Y_g - Y_g^\top Y_g) \\ &= \frac{1}{n_g}(\|X_g\beta - Y_g\|_2^2 - Y_g^\top Y_g), \end{aligned}$$

is strongly convex in β as long as X_g has full rank, thus l_ζ is strongly convex.

Finally

$$\nabla h_\zeta(\beta) = \nabla \frac{\log(\sum_{g=1}^G e^{\zeta h_g(\beta)})}{\zeta} = \frac{\sum_{g=1}^G e^{\zeta h_g(\beta)} \nabla h_g(\beta)}{e^{h_\zeta}}.$$

□

Proof of Proposition 2. If we can show that Assumptions A.1 from [3] holds for the soft maximin problem (7) we can use Theorem A.1 in [3] (or Lemma 4 in [12]) to establish that the iterates have an accumulation point. Theorem 1 in [12] then establishes this accumulation point as a critical point for F.

Let $\Delta > 0$, $\beta_0 \in \mathbb{R}^p$, and define the set

$$\begin{aligned} A_0 &:= \{\beta : F(\beta) \leq F(\beta_0)\} \\ B_{0,\Delta} &:= \{\beta : \|\beta - \beta'\| < \Delta, \beta' \in A_0\}. \end{aligned}$$

A.1(i): $\tilde{l}$ is ν-strongly convex by Proposition 1 implying A_0 is compact hence B_0 is compact as a closed neighborhood of A_0. As $\tilde{l}$ is C^∞ everywhere, $\nabla\tilde{l}$ is Lipschitz on B_0.

A.1(ii): Is satisfied by assumption

A.1(iii): Clearly $F \geq 0$. Furthermore F is continuous hence uniformly continuous on the compact set A_0.

A.1(iv) $\sup_{\beta\in A_0} \|\nabla l\| < \infty$ as A_0 is compact and ∇l is continuous. $\sup_{\beta\in A_0} \|J\| < \infty$ as A_0 is compact and J is continuous. Finally also $\inf J = 0$. □

Appendix B. Multi component tensor array structure

Here we introduce a set of structural model assumptions, for the model framework in subsection 2.1, that can be exploited to obtain a matrix free

and computationally efficient version of algorithm 1. We will call it the multi component array-tensor (MCAT) structure.

In the MCAT framework the gth response vector y_g is given as $y_g = \mathrm{vec}(\boldsymbol{Y}_g)$ (the vec operator produce one long column vector), where $\boldsymbol{Y}_g$ is the $n_1 \times \cdots \times n_d$ d-dimensional response array. Furthermore the random design matrix X_g is assumed to be a concatenation of c matrices

$$X_g = [X_{g,1}|X_{g,2}|\ldots|X_{g,c}],$$

where the rth component is a tensor product,

$$X_{g,r} = X_{g,r,d} \otimes X_{g,r,d-1} \otimes \cdots \otimes X_{g,r,1}, \tag{B.1}$$

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

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

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

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

$$B_g^\top = (\mathrm{vec}(\boldsymbol{B}_{g,1})^\top, \ldots, \mathrm{vec}(\boldsymbol{B}_{g,c})^\top),$$

with $\boldsymbol{B}_{g,r}$ a $p_{r,1} \times \cdots \times p_{r,d}$ d-dimensional array. We let $\langle \boldsymbol{B}_{g,r} \rangle := \langle \boldsymbol{B}_{g,1}, \ldots, \boldsymbol{B}_{g,c} \rangle$ denote the tuple of parameter arrays.

Given this structure it is possible to define a map, ρ, such that with $B_{g,r} = \mathrm{vec}(\boldsymbol{B}_{g,r})$,

$$X_{g,r} B_{g,r} = \mathrm{vec}\big(\rho(X_{g,r,d}, \ldots, \rho(X_{g,r,2}, (\rho(X_{g,r,1}, \boldsymbol{B}_{g,r}))) \ldots)\big) \tag{B.2}$$

for $r = 1, \ldots, c$. The algebraic details of ρ are spelled out in [11]. The identity (B.2) provides the sole justification for the MCAT framework.

Appendix C. Proximity operator

The operator $p_{\delta g} : \mathbb{R}^p \to \mathbb{R}^p$ defined by

$$p_{\delta g}(y) := \underset{x \in \mathbb{R}^p}{\arg\min} \Big\{ \frac{1}{2\delta} \|x - y\|_2^2 + g(x) \Big\}, \tag{C.1}$$

for any proper convex function g and $\delta > 0$ is the so called proximity operator introduced in [13] and in [14].

[1] N. Meinshausen, P. Bühlmann, Maximin effects in inhomogeneous large-scale data, The Annals of Statistics 43 (4) (2015) 1801–1830.

[2] A. Lund, M. Vincent, N. R. Hansen, Penalized estimation in large-scale generalized linear array models, Journal of Computational and Graphical Statistics 0 (ja) (0) 0–0. doi:10.1080/10618600.2017.1279548.

[3] X. Chen, Z. Lu, T. K. Pong, Penalty methods for a class of non-lipschitz optimization problems, SIAM Journal on Optimization 26 (3) (2016) 1465–1492.

[4] A. Lund, SMMA: Soft Maximin Estimation for Large Scale Array-Tensor Models, r package version 1.0.1 (2017).
URL https://CRAN.R-project.org/package=SMMA

[5] P. E. Roland, A. Hanazawa, C. Undeman, D. Eriksson, T. Tompa, H. Nakamura, S. Valentiniene, B. Ahmed, Cortical feedback depolarization waves: A mechanism of top-down influence on early visual areas, Proceedings of the National Academy of Sciences 103 (33) (2006) 12586–12591.

[6] A. Grinvald, T. Bonhoeffer, Optical imaging of electrical activity based on intrinsic signals and on voltage sensitive dyes: The methodology (2002).
URL http://cnc.cj.uc.pt/BEB/private/pdfs/GenePercep/KerstinSchmidt/opticalmethodology.pdf

[7] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society. Series B (Methodological) 58 (1) (1996) 267–288.
URL http://www.jstor.org/stable/2346178

[8] P. Tseng, S. Yun, A coordinate gradient descent method for nonsmooth separable minimization, Mathematical Programming 117 (1-2) (2009) 387–423.

[9] J. Roll, Piecewise linear solution paths with application to direct weight optimization, Automatica 44 (11) (2008) 2732–2737.

[10] J. Friedman, T. Hastie, R. Tibshirani, Regularization paths for generalized linear models via coordinate descent, Journal of statistical software 33 (1) (2010) 1.

[11] I. D. Currie, M. Durban, P. H. Eilers, Generalized linear array models with applications to multidimensional smoothing, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68 (2) (2006) 259–280.

[12] S. J. Wright, R. D. Nowak, M. A. Figueiredo, Sparse reconstruction by separable approximation, IEEE Transactions on Signal Processing 57 (7) (2009) 2479–2493.

[13] J.-J. Moreau, Fonctions convexes duales et points proximaux dans un espace hilbertian., C. R. Acad. Sci., Paris 255 (1962) 2897–2899.

[14] G. J. Minty, Monotone (nonlinear) operators in hilbert space, Duke Math. J. 29 (3) (1962) 341–346.

Part III

Software

Chapter 8

The glamlasso R-package

Lund, A. (2016). glamlasso: Penalization in Large Scale Generalized Linear Array Models. R package version 2.0.1. URL `https://CRAN.R-project.org/package=glamlasso`.

Package 'glamlasso'

August 19, 2016

Type Package

Title Penalization in Large Scale Generalized Linear Array Models

Version 2.0.1

Date 2016-08-01

Author Adam Lund

Maintainer Adam Lund <adam.lund@math.ku.dk>

Description Efficient design matrix free procedure for penalized estimation in large scale 2 and 3-dimensional generalized linear array models. Currently either Lasso or SCAD penalized estimation is possible for the followings models: 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.

License GPL-3

Imports Rcpp (>= 0.11.2)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-08-19 17:05:02

R topics documented:

`glamlasso` *Penalization in Large Scale Generalized Linear Array Models*

Description

Efficient design matrix free procedure for fitting large scale penalized 2 or 3-dimensional generalized linear array models. Currently the LASSO penalty and the SCAD penalty are both implemented. Furthermore, 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 is currently implemented. The `glamlasso` function utilize a gradient descent and proximal gradient based algorithm, see *Lund et al., 2015*.

Usage

```
glamlasso(X,
          Y,
          family = c("gaussian", "binomial", "poisson", "gamma"),
          penalty = c("lasso", "scad"),
          Weights = NULL,
          nlambda = 100,
          lambda.min.ratio = 1e-04,
          lambda = NULL,
          penalty.factor = NULL,
          reltolinner = 1e-07,
          reltolouter = 1e-04,
          maxiter = 15000,
          steps = 1,
          maxiterinner = 3000,
          maxiterouter = 25,
          btinnermax = 100,
          iwls = c("exact", "identity", "kron1", "kron2"),
          nu = 1)
```

Arguments

`X`	A list containing the Kronecker components (2 or 3) of the Kronecker design matrix. These are matrices of sizes $n_i \times p_i$.
`Y`	The response values, an array of size $n_1 \times \cdots \times n_d$. For option `family = "binomial"` this array must contain the proportion of successes and the number of trials is then specified as `Weights` (see below).
`family`	A string specifying the model family (essentially the response distribution). Possible values are `"gaussian"`, `"binomial"`, `"poisson"`, `"gamma"`.
`penalty`	A string specifying the penalty. Possible values are `"lasso"`, `"scad"`.
`Weights`	Observation weights, an array of size $n_1 \times \cdots \times n_d$. For option `family = "binomial"` this array must contain the number of trials and must be provided.

`nlambda` The number of `lambda` values.

`lambda.min.ratio`
The smallest value for `lambda`, given as a fraction of λ_{max}; the (data derived) smallest value for which all coefficients are zero.

`lambda` The sequence of penalty parameters for the regularization path.

`penalty.factor` An array of size $p_1 \times \cdots \times p_d$. Is multiplied with each element in `lambda` to allow differential shrinkage on the coefficients.

`reltolinner` The convergence tolerance for the inner loop

`reltolouter` The convergence tolerance for the outer loop.

`maxiter` The maximum number of inner iterations allowed for each `lambda` value, when summing over all outer iterations for said `lambda`.

`steps` The number of steps used in the multi-step adaptive lasso algorithm for non-convex penalties. Automatically set to 1 when `penalty = "lasso"`.

`maxiterinner` The maximum number of inner iterations allowed for each outer iteration.

`maxiterouter` The maximum number of outer iterations allowed for each lambda.

`btinnermax` Maximum number of backtracking steps allowed in each inner iteration. Default is `btinnermax = 100`.

`iwls` A string indicating whether to use the exact iwls weight matrix or use a kronecker structured approximation to it.

`nu` A number between 0 and 1 that controls the step size δ in the proximal algorithm (inner loop) by scaling the upper bound $\hat{L}_h$ on the Lipschitz constant L_h (see *Lund et al., 2015*). For `nu = 1` backtracking never occurs and the proximal step size is always $\delta = 1/\hat{L}_h$. For `nu = 0` backtracking always occurs and the proximal step size is initially $\delta = 1$. For `0 < nu < 1` the proximal step size is initially $\delta = 1/(\nu\hat{L}_h)$ and backtracking is only employed if the objective function does not decrease. A `nu` close to 0 gives large step sizes and presumably more backtracking in the inner loop. The default is `nu = 1` and the option is only used if `iwls = "exact"`.

Details

We consider a generalized linear model (GLM) with Kronecker structured design matrix given as

$$X = \bigotimes_{i=1}^{d} X_i,$$

where $X_1, X_2, \ldots$ are the marginal $n_i \times p_i$ design matrices (Kronecker components).

We use the generalized linear array model (GLAM) framework to write the model equation as

$$g(M) = \rho(X_d, \rho(X_{d-1}, \ldots, \rho(X_1, \Theta))),$$

where ρ is the so called rotated H-transfrom, M is an array containing the mean of the response variable array Y, Θ is the model coefficient (parameter) array and g is a link function. See *Currie et al., 2006* for more details.

Let $\theta := vec(\Theta)$ denote the vectorized version of the parameter array. The related log-likelihood is a function of θ through the linear predictor η i.e. $\theta \mapsto l(\eta(\theta))$. In the usual exponential family framework this can be expressed as

$$l(\eta(\theta)) = \sum_{i=1}^{n} a_i \frac{y_i \vartheta(\eta_i(\theta)) - b(\vartheta(\eta_i(\theta)))}{\psi} + c(y_i, \psi)$$

where ϑ, the canonical parameter map, is linked to the linear predictor via the identity $\eta(\theta) = g(b'(\vartheta))$ with b the cumulant function. Here $a_i \geq 0, i = 1, \ldots, n$ are observation weights and ψ is the dispersion parameter.

Using only the marginal matrices $X_1, X_2, \ldots$ and with J a penalty function, the function `glamlasso` solves the penalized estimation problem

$$\min_{\theta} -l(\eta(\theta)) + \lambda J(\theta),$$

in the GLAM setup for a sequence of penalty parameters $\lambda > 0$. The underlying algorithm is based on an outer gradient descent loop and an inner proximal gradient based loop. We note that if J is not convex, as with the SCAD penalty, we use the multiple step adaptive lasso procedure to loop over the inner proximal algorithm, see *Lund et al., 2015* for more details.

Value

An object with S3 Class "glamlasso".

`family`	A string indicating the model family.
`coef`	A $p_1 \cdots p_d \times$ `nlambda` matrix containing the estimates of the model coefficients (`beta`) for each `lambda`-value.
`lambda`	A vector containing the sequence of penalty values used in the estimation procedure.
`df`	The number of nonzero coefficients for each value of `lambda`.
`dimcoef`	A vector giving the dimension of the model coefficient array β.
`dimobs`	A vector giving the dimension of the observation (response) array `Y`.
`Iter`	A list with 4 items: `bt_iter_inner` is total number of backtracking steps performed in the inner loop, `bt_enter_inner` is the number of times the backtracking is initiated in the inner loop, `bt_iter_outer` is total number of backtracking steps performed in the outer loop, and `iter_mat` is a `nlambda` $\times$ `maxiterouter` matrix containing the number of inner iterations for each `lambda` value and each outer iteration and `iter` is total number of iterations i.e. `sum(Iter)`.

Author(s)

Adam Lund

Maintainer: Adam Lund, <adam.lund@math.ku.dk>

References

Lund, A., M. Vincent, and N. R. Hansen (2015). Penalized estimation in large-scale generalized linear array models. *ArXiv*.

Currie, I. D., M. Durban, and P. H. C. Eilers (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society. Series B*. 68, 259-280.

Examples

```
## Not run:
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), n3, p3)
X <- list(X1, X2, X3)

##gaussian example
Beta <- array(rnorm(p1 * p2 * p3) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, mu), dim = c(n1, n2, n3))

fit <- glamlasso(X, Y, family = "gaussian", penalty = "lasso", iwls = "exact")
Betafit <- fit$coef

modelno <- length(fit$lambda)
m <- min(Betafit[ , modelno], c(Beta))
M <- max(Betafit[ , modelno], c(Beta))
plot(c(Beta), type="l", ylim = c(m, M))
lines(Betafit[ , modelno], col = "red")

##poisson example
Beta <- array(rnorm(p1 * p2 * p3, 0, 0.1) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))

mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rpois(n1 * n2 * n3, exp(mu)), dim = c(n1, n2, n3))
fit <- glamlasso(X, Y, family = "poisson", penalty = "lasso", iwls = "exact", nu = 0.1)
Betafit <- fit$coef

modelno <- length(fit$lambda)
m <- min(Betafit[ , modelno], c(Beta))
M <- max(Betafit[ , modelno], c(Beta))
plot(c(Beta), type="l", ylim = c(m, M))
lines(Betafit[ , modelno], col = "red")

## End(Not run)
```

`objective` *Compute objective values*

Description

Computes the objective values of the penalized log-likelihood problem for the models implemented in the package glamlasso.

Usage

```
objective(Y,
          Weights,
          X,
          Beta,
          lambda,
          penalty.factor,
          family,
          penalty)
```

Arguments

`Y`	The response values, an array of size $n_1 \times \cdots \times n_d$.
`Weights`	Observation weights, an array of size $n_1 \times \cdots \times n_d$.
`X`	A list containing the tensor components of the tensor design matrix, each of size $n_i \times p_i$.
`Beta`	A coefficient matrix of size $p_1 \cdots p_d \times$ `nlambda`.
`lambda`	The sequence of penalty parameters for the regularization path.
`penalty.factor`	An array of size $p_1 \times \cdots \times p_d$. Is multiplied with each element in `lambda` to allow differential shrinkage on the coefficients.
`family`	A string specifying the model family (essentially the response distribution).
`penalty`	A string specifying the penalty.

Value

A vector of length `length(lambda)` containing the objective values for each `lambda` value.

Examples

```
## Not run:
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), n3, p3)
Beta <- array(rnorm(p1 * p2 * p3) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, mu), dim = c(n1, n2, n3))
```

```
fit <- glamlasso(list(X1, X2, X3), Y, family = "gaussian", penalty = "lasso", iwls = "exact")
objfit <- objective(Y, NULL, list(X1, X2, X3), fit$coef, fit$lambda, NULL, fit$family)
plot(objfit, type = "l")

## End(Not run)
```

`predict.glamlasso` *Make Prediction From a glamlasso Object*

Description

Given new covariate data this function computes the linear predictors based on the estimated model coefficients in an object produced by the function `glamlasso`. Note that the data can be supplied in two different formats: i) as a $n' \times p$ matrix (p is the number of model coefficients and n' is the number of new data points) or ii) as a list of two or three matrices each of size $n_i' \times p_i, i = 1, 2, 3$ (n_i' is the number of new marginal data points in the ith dimension).

Usage

```
## S3 method for class 'glamlasso'
predict(object, x = NULL, X = NULL, ...)
```

Arguments

`object`	An object of Class glamlasso, produced with `glamlasso`.
`x`	a matrix of size $n' \times p$ with n' is the number of new data points.
`X`	A list containing the data matrices each of size $n_i' \times p_i$, where n_i' is the number of new data points in the ith dimension.
`...`	ignored

Value

A list of length `nlambda` containing the linear predictors for each model. If new covariate data is supplied in one $n' \times p$ matrix `x` each item is a vector of length n'. If the data is supplied as a list of matrices each of size $n_i' \times p_i$, each item is an array of size $n_1' \times \cdots \times n_d'$, with $d \in \{2, 3\}$.

Author(s)

Adam Lund

Examples

```
## Not run:
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), n3, p3)
```

```
Beta <- array(rnorm(p1 * p2 * p3) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, mu), dim = c(n1, n2, n3))
fit <- glamlasso(list(X1, X2, X3), Y, family = "gaussian", penalty = "lasso", iwls = "exact")

##new data in matrix form
x <- matrix(rnorm(p1 * p2 * p3), nrow = 1)
predict(fit, x = x)[[100]]

##new data in tensor component form
X1 <- matrix(rnorm(p1), nrow = 1)
X2 <- matrix(rnorm(p2), nrow = 1)
X3 <- matrix(rnorm(p3), nrow = 1)
predict(fit, list(X1, X2, X3))[[100]]

## End(Not run)
```

`print.glamlasso` *Print Function for objects of Class glamlasso*

Description

This function will print some information about the glamlasso object.

Usage

```
## S3 method for class 'glamlasso'
print(x, ...)
```

Arguments

`x`	A glamlasso object
`...`	ignored

Author(s)

Adam Lund

Examples

```
## Not run:
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), n3, p3)
Beta <- array(rnorm(p1 * p2 * p3) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
mu <- RH(X3, RH(X2, RH(X1, Beta)))
```

```
Y <- array(rnorm(n1 * n2 * n3, mu), dim = c(n1, n2, n3))
fit <- glamlasso(list(X1, X2, X3), Y, family = "gaussian", penalty = "lasso", iwls = "exact")
fit

## End(Not run)
```

RH *The Rotated H-transform of a 3d Array by a Matrix*

Description

This function is an implementation of the ρ-operator found in *Currie et al 2006*. It forms the basis of the GLAM arithmetic.

Usage

```
RH(M, A)
```

Arguments

M	a $n \times p_1$ matrix.
A	a 3d array of size $p_1 \times p_2 \times p_3$.

Details

For details see *Currie et al 2006*. Note that this particular implementation is not used in the optimization routines underlying the glamlasso procedure.

Value

A 3d array of size $p_2 \times p_3 \times n$.

Author(s)

Adam Lund

References

Currie, I. D., M. Durban, and P. H. C. Eilers (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society. Series B*. 68, 259-280.

Index

Chapter 9

The SMMA R-package

Lund, A. (2017). S MMA: Soft Maximin Estimation for Large Scale Array-Tensor Models. R package version 1.0.1. URL `https://CRAN.R-project.org/package=SMMA`.

Package 'SMMA'

March 30, 2017

Type Package

Title Soft Maximin Estimation for Large Scale Array-Tensor Models

Version 1.0.1

Date 2017-03-29

Author Adam Lund

Maintainer Adam Lund <adam.lund@math.ku.dk>

Description
Efficient design matrix free procedure for solving a soft maximin problem for large scale array-tensor structured models. Currently Lasso and SCAD penalized estimation is implemented.

License GPL-3

Imports Rcpp (>= 0.11.2)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-03-30 11:28:41 UTC

R topics documented:

`predict.SMMA` *Make Prediction From a SMMA Object*

Description

Given new covariate data this function computes the linear predictors based on the estimated model coefficients in an object produced by the function `softmaximin`. Note that the data can be supplied in two different formats: i) as a $n' \times p$ matrix (p is the number of model coefficients and n' is the number of new data points) or ii) as a list of two or three matrices each of size $n'_i \times p_i, i = 1, 2, 3$ (n'_i is the number of new marginal data points in the ith dimension).

Usage

```
## S3 method for class 'SMMA'
predict(object, x = NULL, X = NULL, ...)
```

Arguments

`object`	An object of class SMMA, produced with `softmaximin`
`x`	a matrix of size $n' \times p$ with n' is the number of new data points.
`X`	a list containing the data matrices each of size $n'_i \times p_i$, where n'_i is the number of new data points in the ith dimension.
`...`	ignored

Value

A list of length `nlambda` containing the linear predictors for each model. If new covariate data is supplied in one $n' \times p$ matrix `x` each item is a vector of length n'. If the data is supplied as a list of matrices each of size $n'_i \times p_i$, each item is an array of size $n'_1 \times \cdots \times n'_d$, with $d \in \{2, 3\}$.

Author(s)

Adam Lund

Examples

```
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1, 0, 0.5), n1, p1)
X2 <- matrix(rnorm(n2 * p2, 0, 0.5), n2, p2)
X3 <- matrix(rnorm(n3 * p3, 0, 0.5), n3, p3)
X <- list(X1, X2, X3)

component <- rbinom(p1 * p2 * p3, 1, 0.1)
```

```
Beta1 <- array(rnorm(p1 * p2 * p3, 0, .1) + component, c(p1 , p2, p3))
Beta2 <- array(rnorm(p1 * p2 * p3, 0, .1) + component, c(p1 , p2, p3))
mu1 <- RH(X3, RH(X2, RH(X1, Beta1)))
mu2 <- RH(X3, RH(X2, RH(X1, Beta2)))
Y1 <- array(rnorm(n1 * n2 * n3, mu1), dim = c(n1, n2, n3))
Y2 <- array(rnorm(n1 * n2 * n3, mu2), dim = c(n1, n2, n3))

Y <- array(NA, c(dim(Y1), 2))
Y[,,, 1] <- Y1; Y[,,, 2] <- Y2;

fit <- softmaximin(X, Y, penalty = "lasso", alg = "npg")

##new data in matrix form
x <- matrix(rnorm(p1 * p2 * p3), nrow = 1)
predict(fit, x = x)[[15]]

##new data in tensor component form
X1 <- matrix(rnorm(p1), nrow = 1)
X2 <- matrix(rnorm(p2), nrow = 1)
X3 <- matrix(rnorm(p3), nrow = 1)
predict(fit, X = list(X1, X2, X3))[[15]]
```

`print.SMMA` *Print Function for objects of Class SMMA*

Description

This function will print some information about the SMMA object.

Usage

```
## S3 method for class 'SMMA'
print(x, ...)
```

Arguments

`x`	a SMMA object
`...`	ignored

Author(s)

Adam Lund

Examples

```
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1, 0, 0.5), n1, p1)
X2 <- matrix(rnorm(n2 * p2, 0, 0.5), n2, p2)
X3 <- matrix(rnorm(n3 * p3, 0, 0.5), n3, p3)
X <- list(X1, X2, X3)

component <- rbinom(p1 * p2 * p3, 1, 0.1)
Beta1 <- array(rnorm(p1 * p2 * p3, 0, .1) + component, c(p1 , p2, p3))
Beta2 <- array(rnorm(p1 * p2 * p3, 0, .1) + component, c(p1 , p2, p3))
mu1 <- RH(X3, RH(X2, RH(X1, Beta1)))
mu2 <- RH(X3, RH(X2, RH(X1, Beta2)))
Y1 <- array(rnorm(n1 * n2 * n3, mu1), dim = c(n1, n2, n3))
Y2 <- array(rnorm(n1 * n2 * n3, mu2), dim = c(n1, n2, n3))

Y <- array(NA, c(dim(Y1), 2))
Y[,,, 1] <- Y1; Y[,,, 2] <- Y2;

fit <- softmaximin(X, Y, penalty = "lasso", alg = "npg")
fit
```

RH *The Rotated H-transform of a 3d Array by a Matrix*

Description

This function is an implementation of the ρ-operator found in *Currie et al 2006.* It forms the basis of the GLAM arithmetic.

Usage

```
RH(M, A)
```

Arguments

M	a $n \times p_1$ matrix.
A	a 3d array of size $p_1 \times p_2 \times p_3$.

Details

For details see *Currie et al 2006.* Note that this particular implementation is not used in the optimization routines underlying the glamlasso procedure.

Value

A 3d array of size $p_2 \times p_3 \times n$.

Author(s)

Adam Lund

References

Currie, I. D., M. Durban, and P. H. C. Eilers (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society. Series B*. 68, 259-280.

Examples

```
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), n3, p3)

Beta <- array(rnorm(p1 * p2 * p3, 0, 1), c(p1 , p2, p3))
max(abs(c(RH(X3, RH(X2, RH(X1, Beta)))) - kronecker(X3, kronecker(X2, X1)) %*% c(Beta)))
```

SMMA *Maximin Estimation for Large Scale Array Data with Known Groups*

Description

Efficient design matrix free procedure for solving a soft maximin problem for large scale array-tensor structured models, see *Lund et al., 2017*. Currently Lasso and SCAD penalized estimation is implemented.

Usage

```
softmaximin(X,
            Y,
            penalty = c("lasso", "scad"),
            nlambda = 30,
            lambda.min.ratio = 1e-04,
            lambda = NULL,
            penalty.factor = NULL,
            reltol = 1e-05,
            maxiter = 15000,
            steps = 1,
```

```
            btmax = 100,
            zeta = 2,
            c = 0.001,
            Delta0 = 1,
            nu = 1,
            alg = c("npg", "mfista"),
            log = TRUE)
```

Arguments

X	A list containing the Kronecker components (2 or 3) of the Kronecker design matrix. These are matrices of sizes $n_i \times p_i$.
Y	The response values, an array of size $n_1 \times \cdots \times n_d \times G$.
penalty	A string specifying the penalty. Possible values are "lasso", "scad".
nlambda	The number of lambda values.
lambda.min.ratio	The smallest value for lambda, given as a fraction of λ_{max}; the (data dependent) smallest value for which all coefficients are zero.
lambda	The sequence of penalty parameters for the regularization path.
penalty.factor	An array of size $p_1 \times \cdots \times p_d$. Is multiplied with each element in lambda to allow differential shrinkage on the coefficients.
reltol	The convergence tolerance for the inner loop.
maxiter	The maximum number of iterations allowed for each lambda value, when summing over all outer iterations for said lambda.
steps	The number of steps used in the multi-step adaptive lasso algorithm for non-convex penalties. Automatically set to 1 when penalty = "lasso".
btmax	Maximum number of backtracking steps allowed in each iteration. Default is btmax = 100.
zeta	Constant controlling the softmax apprximation accuracy. Must be strictly positive. Default is zeta = 2.
c	constant used in the NPG algorithm. Must be strictly positive. Default is c = 0.001.
Delta0	constant used to bound the stepsize. Must be strictly positive. Default is Delta0 = 1.
nu	constant used to control the stepsize. Must be positive. A small value gives a big stepsize. Default is nu = 1.
alg	string indicating which algortihm to use. Possible values are "npg", "mfista".
log	logical variable indicating wheter to use log-loss to or not. TRUE is default and yields the problem described below.

Details

We consider the mixed model setup from *Meinshausen and Buhlmann, 2015* for array data with a known fixed group structure and tensor structured design matrix.

For $g \in \{1, \ldots, G\}$ let n be the number of observations in each group. With $Y_g := (y_i, \ldots, y_{i_n})^\top$ the group-specific $n_1 \times \cdots \times n_d$ response array and $X := (x_i \mid \ldots \mid x_{i_n})^\top$ a $n \times p$ design matrix, with tensor structure

$$X = \bigotimes_{i=1}^{d} X_i,$$

where for $d = 2, 3$, $X_1, \ldots, X_d$ are the marginal $n_i \times p_i$ design matrices (Kronecker components).

We use the array model framework, see *Currie et al., 2006*, to write the model equation as

$$Y_g = \rho(X_d, \rho(X_{d-1}, \ldots, \rho(X_1, B_g))) + E,$$

where ρ is the so called rotated H-transfrom, B_g for each g is a random $p_1 \times \cdots \times p_d$ parameter array and E is $n_1 \times \cdots \times n_d$ error array uncorrelated with X.

In *Meinshausen and Buhlmann, 2015* it is suggested to maximize the minimal empirical explained variance

$$\hat{V}_g(\beta) := \frac{1}{n}(2\beta^\top X^\top y_g - \beta^\top X^\top X \beta),$$

where $y_g := vec(Y_g)$. In *Lund et al., 2017* a soft version of this problem, the soft maximin problem, given as

$$\min_\beta \log\left(\sum_{g=1}^{G} \exp(-\zeta \hat{V}_g(\beta))\right) + \lambda J(\beta),$$

is suggested, for J a proper and convex function and $\zeta > 0$.

For $d = 2, 3$ and using only the marginal matrices $X_1, X_2, \ldots$, the function `softmaximin` solves the soft maximin problem for a sequence of penalty parameters $\lambda_{max} > \ldots > \lambda_{min} > 0$. The underlying algorithm is based on a non-monotone proximal gradient method. We note that if J is not convex, as with the SCAD penalty, we use the multiple step adaptive lasso procedure to loop over the proximal algorithm, see *Lund et al., 2017* for more details.

Value

An object with S3 Class "SMMA".

`spec`	A string indicating the array dimension (2 or 3) and the penalty.
`coef`	A $p_1 \cdots p_d \times$ `nlambda` matrix containing the estimates of the model coefficients (`beta`) for each `lambda`-value.
`lambda`	A vector containing the sequence of penalty values used in the estimation procedure.
`Obj`	A matrix containing the objective values for each iteration and each model.
`df`	The number of nonzero coefficients for each value of `lambda`.
`dimcoef`	A vector giving the dimension of the model coefficient array β.
`dimobs`	A vector giving the dimension of the observation (response) array `Y`.
`Iter`	A list with 4 items: `bt_iter` is total number of backtracking steps performed, `bt_enter` is the number of times the backtracking is initiated, and `iter_mat` is a vector containing the number of iterations for each `lambda` value and `iter` is total number of iterations i.e. `sum(Iter)`.

Author(s)

Adam Lund

Maintainer: Adam Lund, <adam.lund@math.ku.dk>

References

Lund, A., S. W. Mogensen and N. R. Hansen (2017). Estimating Soft Maximin Effects in Heterogeneous Large-scale Array Data. *Preprint*.

Meinshausen, N and P. Buhlmann (2015). Maximin effects in inhomogeneous large-scale data. *The Annals of Statistics*. 43, 4, 1801-1830.

Currie, I. D., M. Durban, and P. H. C. Eilers (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society. Series B*. 68, 259-280.

Examples

```
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), n3, p3)
X <- list(X1, X2, X3)

component <- rbinom(p1 * p2 * p3, 1, 0.1)
Beta1 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu1 <- RH(X3, RH(X2, RH(X1, Beta1)))
Y1 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu1
Beta2 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu2 <- RH(X3, RH(X2, RH(X1, Beta2)))
Y2 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu2
Beta3 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu3 <- RH(X3, RH(X2, RH(X1, Beta3)))
Y3 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu3
Beta4 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu4 <- RH(X3, RH(X2, RH(X1, Beta4)))
Y4 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu4
Beta5 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu5 <- RH(X3, RH(X2, RH(X1, Beta5)))
Y5 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu5

Y <- array(NA, c(dim(Y1), 5))
Y[,,, 1] <- Y1; Y[,,, 2] <- Y2; Y[,,, 3] <- Y3; Y[,,, 4] <- Y4; Y[,,, 5] <- Y5;

fit <- softmaximin(X, Y, penalty = "lasso", alg = "npg")
Betafit <- fit$coef

modelno <- 15
```

```
m <- min(Betafit[ , modelno], c(component))
M <- max(Betafit[ , modelno], c(component))
plot(c(component), type="l", ylim = c(m, M))
lines(Betafit[ , modelno], col = "red")
```

Index

Bibliography

Adler, R. J. and J. E. Taylor (2009). *Random fields and geometry*. Springer Science & Business Media.

Ahmed, B., A. Hanazawa, C. Undeman, D. Eriksson, S. Valentiniene, and P. E. Roland (2008). Cortical dynamics subserving visual apparent motion. *Cerebral Cortex 18*, 2796–2810.

Beck, A. and M. Teboulle (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences 2*(1), 183–202.

Bertsekas, D. P. (1999). *Nonlinear programming*. Athena scientific Belmont.

Buis, P. E. and W. R. Dyksen (1996). Efficient vector and parallel manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS) 22*(1), 18–23.

Chemla, S. and F. Chavane (2010). Voltage-sensitive dye imaging: technique review and models. *Journal of Physiology-Paris 104*(1), 40–50.

Chen, X., Z. Lu, and T. K. Pong (2016). Penalty methods for a class of non-lipschitz optimization problems. *SIAM Journal on Optimization 26*(3), 1465–1492.

Cox, S. G. (2012). Stochastic differential equations in banach spaces: Decoupling, delay equations, and approximations in space and time. *PhD thesis*.

Currie, I. D., M. Durban, and P. H. Eilers (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68*(2), 259–280.

Da Prato, G. and J. Zabczyk (2014). *Stochastic equations in infinite dimensions*. Cambridge university press.

Dawson, D. A. and H. Salehi (1980). Spatially homogeneous random evolutions. *Journal of Multivariate Analysis 10*(2), 141–180.

De Boor, C. (1979). Efficient computer manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS) 5*(2), 173–182.

Eriksson, D., T. Tompa, and P. E. Roland (2008). Non-linear population firing rates and voltage sensitive dye signals in visual areas 17 and 18 to short duration stimuli. *PLoS One 3*(7), e2673.

Friedman, J., T. Hastie, and R. Tibshirani (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software 33*(1), 1.

Harvey, M. and P. Roland (2013). Laminar firing and membrane dynamics in four visual areas exposed to two objects moving to occlusion. *Frontiers in Systems Neuroscience 7*, 23.

Harvey, M. A., S. Valentiniene, and P. E. Roland (2009). Cortical membrane potential dynamics and laminar firing during object motion. *Frontiers in systems neuroscience 3*, 7.

Hastie, T., R. Tibshirani, and M. Wainwright (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. CRC Press.

Jin, W., R.-J. Zhang, and J.-y. Wu (2002). Voltage-sensitive dye imaging of population neuronal activity in cortical tissue. *Journal of neuroscience methods 115*(1), 13–27.

Krasnoselsky, M. (1955). Two observations about the method of successive approximations. *Usp. Mat. Nauk* (10), 123–127.

Lindquist, M. A. (2008). The statistical analysis of fmri data. *Statistical Science*, 439–464.

Lund, A. (2016). *glamlasso: Penalization in Large Scale Generalized Linear Array Models*. R package version 2.0.1.

Lund, A. (2017). *SMMA: Soft Maximin Estimation for Large Scale Array-Tensor Models*. R package version 1.0.1.

Lund, A. and N. R. Hansen (2017). Sparse network estimation for dynamical spatio-temporal array models. In preparation.

Lund, A., S. W. Mogensen, and N. R. Hansen (2017). Estimating soft maximin effects in large-scale data. In preparation.

Lund, A., M. Vincent, and N. R. Hansen (2017). Penalized estimation in large-scale generalized linear array models. *Journal of Computational and Graphical Statistics*, To appear.

Mann, W. R. (1953). Mean value methods in iteration. *Proceedings of the American Mathematical Society 4*(3), 506–510.

Meinshausen, N. and P. Bühlmann (2015). Maximin effects in inhomogeneous large-scale data. *The Annals of Statistics 43*(4), 1801–1830.

Meyer, Y. (1995). *Wavelets and Operators*. Cambridge Studies in Advanced Mathematics. Cambridge University Press.

Minty, G. J. (1962, 09). Monotone (nonlinear) operators in hilbert space. *Duke Math. J. 29*(3), 341–346.

Mogensen, S. W. (2016). Maximin effects and neuronal activity: Computationally efficient analysis of large-scale heterogeneous data. Master's thesis.

Moreau, J. (1965). Proximité et dualité dans un espace hilbertien. *Bulletin de la Société Mathématique de France 93*, 273–299.

Moreau, J.-J. (1962). Fonctions convexes duales et points proximaux dans un espace hilbertian. *C. R. Acad. Sci., Paris 255*, 2897–2899.

Nemirovsky, A. and D. Yudin (1983). *Problem Complexity and Method Efficiency in Optimization*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady 27*(2), 372–376.

Pereyra, V. and G. Scherer (1973). Efficient computer manipulation of tensor products with applications to multidimensional approximation. *Mathematics of Computation 27*(123), 595–605.

Peszat, S. and J. Zabczyk (1997). Stochastic evolution equations with a spatially homogeneous wiener process. *Stochastic Processes and their Applications 72*(2), 187–204.

Reed, M. and B. Simon (1972). Methods of modern mathematical physics i: Functional analysis. *New York-London*.

Roland, P. E., A. Hanazawa, C. Undeman, D. Eriksson, T. Tompa, H. Nakamura, S. Valentiniene, and B. Ahmed (2006). Cortical feedback depolarization waves: A mechanism of top-down influence on early visual areas. *Proceedings of the National Academy of Sciences 103*(33), 12586–12591.

Rothman, A. J., E. Levina, and J. Zhu (2010). Sparse multivariate regression with covariance estimation. *Journal of Computational and Graphical Statistics 19*(4), 947–962.

Searle, S. R. (1982). Matrix algebra useful for statistics. *New York 1982*.

Tseng, P. and S. Yun (2009). A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming 117*(1-2), 387–423.

Xu, D., X. Wang, and Z. Yang (2012). Existence-uniqueness problems for infinite dimensional stochastic differential equations with delays. *Journal of Applied Analysis and Computation 2*(4), 449–463.

Yates, F. (1937). The design and analysis of factorial experiments. *Technical Communication* (35).