Machine learning based on Hawkes processes and stochastic optimization

Martin Bompaire

université PARIS-SACLAY

NNT : 2019SACLX030

Thèse de doctorat

Machine learning based on Hawkes processes and stochastic optimization

Thèse de doctorat de l'Université Paris-Saclay
préparée à Ecole polytechnique

École doctorale n°574 Mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques appliquées

Thèse présentée et soutenue à Paris, le 5 juillet 2019, par

M. MARTIN BOMPAIRE

Composition du Jury :

M. Alexandre Gramfort Professeur Assistant, INRIA Paris Saclay	Président
M. Julien Mairal Directeur de Recherche, INRIA Grenoble	Rapporteur
M. Niels Richard Hansen Professeur, University of Copenhagen	Rapporteur
M. Guillaume Garrigos Maître de Conférence, Université Paris Diderot (LPSM)	Examinateur
M. Emmanuel Bacry Directeur de Recherche, Université Paris-Dauphine (CEREMADE)	Directeur de thèse
M. Stéphane Gaïffas Professeur, Université Paris Diderot (LPSM)	Co-directeur de thèse

Remerciements

Je tiens en premier lieu à exprimer ma plus profonde gratitude envers mes directeurs de thèse Stéphane Gaïffas et Emmanuel Bacry. Merci pour leur soutien et leur confiance qui m'ont permis de mener à bien ce travail. Merci de m'avoir fait découvrir le monde de la recherche et de m'avoir fait voyager lors de conférences afin de m'ouvrir l'esprit sur les travaux de la communauté. Merci enfin de m'avoir permis de passer beaucoup de temps à travailler l'aspect programmation tout en ayant su me dire de m'arrêter avant que je m'y égare.

Je remercie Niels Hansen et Julien Mairal d'avoir accepté de rapporter ma thèse. Je suis très honoré par leur lecture attentive de ce manuscrit et leur intérêt pour mon travail. Je suis également très honoré qu'Alexandre Gramfort ait accepté d'être président de mon jury de thèse et que Guillaume Garrigos y ait pris part.

Je suis très reconnaissant envers mes co-auteurs : Jean-François pour la partie théorique et Søren et Philip pour la programmation de la librairie `tick`, merci pour leur patience et leur écoute face à ma détermination.

Merci à tous les doctorants de Polytechnique avec qui j'ai passé de très bons moments durant ces trois ans et en particulier : Maryan mon binôme avec qui nous nous suivons depuis l'ENSAE, Alain mon premier co-bureau, mais aussi Marcello et Massil qui m'ont montré la voie, et Yiyang et Peng qui la suivent à leur tour. Merci aussi à toute l'équipe CNAM parmi lesquels Prosper, Youcef, Phong, Dian, Raphaël et Anastasiia et enfin au secrétariat du CMAP pour leur disponibilité et leur gentillesse.

Merci également à Criteo, pour me permettre de continuer à allier la théorie à l'implémentation et pour avoir accueilli ma soutenance de thèse. Tous ceux qui ont évité le trajet vers Palaiseau se joignent à moi pour vous remercier.

Pour finir, un grand merci à mes parents, mes frères, mes belles-soeurs, mes amis, et tout particulièrement à Charlotte qui s'apprête à devenir ma femme, pour leur soutien indéféctible pendant ces trois années et leur présence aujourd'hui.

Résumé

Le fil rouge de cette thèse est l'étude des processus de Hawkes. Ces processus ponctuels décryptent l'inter-causalité qui peut avoir lieu entre plusieurs séries d'événements. Concrètement, ils déterminent l'influence qu'ont les événements d'une série sur les événements futurs de toutes les autres séries. Par exemple, dans le contexte des réseaux sociaux, ils décrivent à quel point l'action d'un utilisateur, tel un Tweet, sera susceptible de déclencher des réactions de la part des autres.

Le premier chapitre est une brève introduction sur les processus ponctuels suivie par un approfondissement sur les processus de Hawkes et en particulier sur les propriétés de la paramétrisation à noyaux exponentiels, la plus communément utilisée. Dans le chapitre suivant, nous introduisons une pénalisation adaptative pour modéliser, avec des processus de Hawkes, la propagation de l'information dans les réseaux sociaux. Cette pénalisation est capable de prendre en compte la connaissance a priori des caractéristiques de ces réseaux, telles que les interactions sparses entre utilisateurs ou la structure de communauté, et de les réfléchir sur le modèle estimé. Notre technique utilise des pénalités pondérées dont les poids sont déterminés par une analyse fine de l'erreur de généralisation.

Ensuite, nous abordons l'optimisation convexe et les progrès réalisés avec les méthodes stochastiques du premier ordre avec réduction de variance. Le quatrième chapitre est dédié à l'adaptation de ces techniques pour optimiser le terme d'attache aux données le plus couramment utilisé avec les processus de Hawkes. En effet, cette fonction ne vérifie pas l'hypothèse de gradient-Lipschitz habituellement utilisée. Ainsi, nous travaillons avec une autre hypothèse de régularité, et obtenons un taux de convergence linéaire pour une version décalée de *Stochastic Dual Coordinate Ascent* qui améliore l'état de l'art. De plus, de telles fonctions comportent beaucoup de contraintes linéaires qui sont fréquemment violées par les algorithmes classiques du premier ordre, mais, dans leur version duale ces contraintes sont beaucoup plus aisées à satisfaire. Ainsi, la robustesse de notre algorithme est d'avantage comparable à celle des méthodes du second ordre dont le coût est prohibitif en grandes dimensions.

Enfin, le dernier chapitre présente une nouvelle bibliothèque d'apprentissage statistique pour Python 3 avec un accent particulier mis sur les modèles temporels. Appelée `tick`, cette bibliothèque repose sur une implémentation en C++ et les algorithmes d'optimisation issus de l'état de l'art pour réaliser des estimations très rapides dans un environnement multi-cœurs. Publiée sur Github, cette bibliothèque a été utilisée tout au long de cette thèse pour effectuer des expériences.

Abstract

The common thread of this thesis is the study of Hawkes processes. These point processes decrypt the cross-causality that occurs across several event series. Namely, they retrieve the influence that the events of one series have on the future events of all series. For example, in the context of social networks, they describe how likely an action of a certain user (such as a Tweet) will trigger reactions from the others.

The first chapter consists in a general introduction on point processes followed by a focus on Hawkes processes and more specifically on the properties of the widely used exponential kernels parametrization. In the following chapter, we introduce an adaptive penalization technique to model, with Hawkes processes, the information propagation on social networks. This penalization is able to take into account the prior knowledge on the social network characteristics, such as the sparse interactions between users or the community structure, to reflect them on the estimated model. Our technique uses data-driven weighted penalties induced by a careful analysis of the generalization error.

Next, we focus on convex optimization and recall the recent progresses made with stochastic first order methods using variance reduction techniques. The fourth chapter is dedicated to an adaptation of these techniques to optimize the most commonly used goodness-of-fit of Hawkes processes. Indeed, this goodness-of-fit does not meet the gradient-Lipschitz assumption that is required by the latest first order methods. Thus, we work under another smoothness assumption, and obtain a linear convergence rate for a shifted version of Stochastic Dual Coordinate Ascent that improves the current state-of-the-art. Besides, such objectives include many linear constraints that are easily violated by classic first order algorithms, but in the Fenchel-dual problem these constraints are easier to deal with. Hence, our algorithm's robustness is comparable to second order methods that are very expensive in high dimensions.

Finally, the last chapter introduces a new statistical learning library for Python 3 with a particular emphasis on time-dependent models, tools for generalized linear models and survival analysis. Called `tick`, this library relies on a C++ implementation and state-of-the-art optimization algorithms to provide very fast computations in a single node multi-core setting. Open-sourced and published on Github, this library has been used all along this thesis to perform benchmarks and experiments.

List of papers being part of this thesis

- E. Bacry, M. Bompaire, S. Gaïffas, and J.-F. Muzy *Sparse and low-rank multivariate Hawkes processes*, in revision in Journal of Machine Learning Research.
- M. Bompaire, S. Gaïffas, E. Bacry *Dual optimization for convex constrained objectives without the gradient-Lipschitz assumption*, submitted to Journal of Machine Learning Research.
- E. Bacry, M. Bompaire, P. Deegan, S. Gaïffas, and S. Poulsen. *tick: a Python Library for Statistical Learning, with an emphasis on Hawkes Processes and Time-Dependent Models*, Journal of Machine Learning Research 18, 214:1-214:5, 2018.

Contents

Introduction

This introduction is a short summary of the following chapters. It gives an overview of the work presented in this thesis and sometimes passes quickly on technical details. These details are duly substantiated in the corresponding chapters.

1 Background on Hawkes processes

Temporal point processes are used to study sequences of events in continuous time. Unlike time series, they do not depend on any predefined time resolution and can study multiple time scales at once. This chapter is a short introduction on temporal point processes with a specific focus on Hawkes processes, further details can be found in [DVJ07].

1.1 Temporal point process

We associate to a set of distinct timestamps $\{t_1, \dots, t_n\}$ occurring in a time interval $[0, T]$, the counting process $N_t = \sum_{t_k} \mathbb{1}_{t_k \leq t}$. This counting process is a random process whose distribution is characterized by an *intensity function* $\lambda(t|\mathcal{F}_t)$ which gives the infinitesimal probability with which an event will arise at time t given the information $\mathcal{F}_t$ available up to (but not including) time t. It writes

$$\lambda(t|\mathcal{F}_t) = \lim_{dt \to 0} \frac{\mathbb{P}(N_{t+dt} - N_t = 1 |\ \mathcal{F}_t)}{dt}.$$

In the most simple case, this intensity is constant and the associated process is called homogeneous Poisson process. It describes a phenomenon with no memory and a constant probability of occurrence.

Goodness-of-fit. We call goodness-of-fit a function telling how well a statistical model fits a set of observations. The most common measure of goodness-of-fit is the likelihood function given by [DVJ07]. For convenience, we rather consider the

opposite of its logarithm as an error measure, given by

$$-\log L(\mathscr{F}_T) = \int_0^T \lambda(s|\mathscr{F}_s)\,\mathrm{d}s - \sum_{k=1}^{N_T} \log\lambda(t_k|\mathscr{F}_{t_k}).$$

Another measure is the least squares error inspired by the empirical risk minimization principle. For point processes it writes (see [RBR10, HRBR15])

$$R(\mathscr{F}_T) = \int_0^T \lambda(s|\mathscr{F}_s)^2\,\mathrm{d}s - 2\sum_{k=1}^{N_T} \lambda(t_k|\mathscr{F}_{t_k}). \tag{1}$$

Assuming that N_t has an unknown ground truth intensity λ^*, $\mathbb{E}[R(\mathscr{F}_T)]$ is minimized by λ^*. This second goodness-of-fit is not as common but is easier to optimize in some cases. Both measures encourage intensities functions with high values at times when events occur and low at any other time.

1.2 Hawkes processes

Hawkes processes [Haw71a, HO74] are temporal point processes in which the intensity depends on the process history with an excitation mechanism. They can be understood as the equivalent of autoregressive time series models (AR) [Ham94] but studied in continuous time. This allows to study cross causality that might occur in one or several events series. They have first been used in seismology [Oga99], then in finance [BDHM13, BMM15] and nowadays have found many new applications including crime prediction [LMBB12, Moh13] or social network information propagation [CS08, BBH12, ZZS13a, YZ13, LSV+16].

Multivariate Hawkes processes model the interactions of $D \geq 1$ point processes with an excitation dynamic encompassed by the auto-regressive structure of the conditional intensities. For each point process $i = 1,\dots,D$ it writes

$$\lambda^i(t|\,\mathscr{F}_t) = \mu^i + \sum_{j=1}^{D} \int_0^t \varphi^{ij}(t-s)\,\mathrm{d}N_s^j.$$

The $\mu^i \geq 0$ are called *baseline* intensities and correspond to the exogenous intensities of events for nodes $i = 1,\dots,D$. The φ^{ij} for $1 \leq i,j \leq D$ are called *kernels*, they quantify in magnitude and over time the influence of past events from node j on the intensity of events from node i. The integral matrix $(\Phi)_{1\leq i,j\leq D} = \int_0^T \varphi^{ij}(t)\,\mathrm{d}t$ denotes the expected number of events of type i directly triggered by an event of type j. A Hawkes process admits a stationarity regime when its spectral radius is lower than one: $\varrho(\Phi) < 1$ [BMM15]. Such processes are easily simulated with the thinning algorithm [Oga81]. Figure 1 shows the realization of a 2-dimensional Hawkes process and highlights the excitation mechanism by exhibiting the kernel functions φ^{ij} and the impact of each event on all nodes intensities.

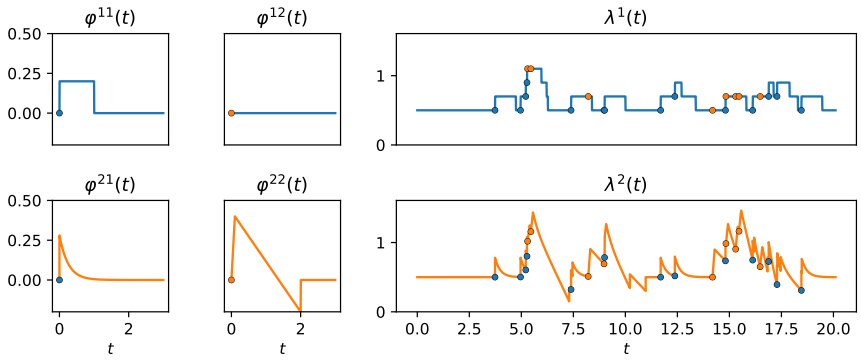

Figure 1: A realization of a 2-dimensional Hawkes process. The 4 kernels are shown on the left hand side. The intensities are displayed on the right hand side (against time, up to time 20), where events are represented by colored dots (blue corresponding to node 1 and orange to node 2).

Estimation. Inferring a Hawkes process consists in estimating its exogenous intensity μ and its kernel functions φ^{ij} either in a parametric or in a non-parametric fashion. In the non-parametric case, the kernels are approximated by histograms in most cases [LM11, ZZS13b, BM14]. This procedure estimates the Hawkes kernels in a very flexible way but is hardly scalable. Recently, [ABG+17] has provided a new non-parametric estimator that infers directly the matrix Φ of kernels' integrals for a better scalability. In the parametric case, the estimators rely on a prior knowledge of the kernel functions which are thus described by a set of parameters. Estimating the kernel functions then revert to estimating these parameters. These procedures are less flexible than non-parametric methods but are more efficient and more robust as the number of parameters to estimate is smaller. Different parametrizations are used to model various behaviors such as delayed response to a triggering event [XFZ16] or a slowly decreasing influence with power-law kernels used in seismology [Oga88] and in finance [BMM15]. However, for scalability reason, most parametric estimators are based on exponentially decaying kernels [ELL11, ZZS13a, TFSZ15, FWR+15, LSK17].

Exponential kernels. The main parametric model for the kernels is the so-called *exponential* kernel, in which we consider $\varphi^{ij}(t) = \alpha^{ij}\beta\exp(-\beta t)$ for $\alpha^{ij} > 0$ and $\beta > 0$ (see kernel φ^{21} in Figure 1). In this model the integral matrix $\Phi = [\alpha^{i,j}]_{1\leq i,j\leq d}$ and $\beta > 0$ is a memory parameter. The couple $(N_t, \lambda(t))$ is a Markov process [BMM15, Proposition 2], and the conditional intensity equation rewrites in a Markovian form

$$\mathrm{d}\lambda^i(t|\,\mathscr{F}_t) = \sum_{j=1}^{D} \beta(\mu^j - \lambda^j(t|\,\mathscr{F}_t))\,\mathrm{d}t + \alpha^{ij}\beta\,\mathrm{d}N_s^j$$

for $i = 1, \dots D$. Hence, instead of requiring to look at all the previous events, the value of the conditional intensity λ^i at a time t_2 can be obtained from its value at a previous time $t_1 < t_2$ and the events that have occurred between t_1 and t_2. This property enables much more efficient computations and a better scaling for both simulation and estimation.

A more general approach is the *sum of exponentials* kernel [LV14], namely $\varphi^{ij}(t) = \sum_{u=1}^{U} \alpha_u^{ij} \beta_u \exp(-\beta_u t)$ for $\alpha_u^{ij} > 0$ and $\beta_u > 0$. These kernels still benefit from the Markov property and generalize better as they deal with several time scales β_u and thus can approximate for example power-law kernels [HBB13, FS15]. For convexity reasons, the memory parameters β are generally fixed during the estimation [LV14]. Then, in order to retrieve the parameters μ and α, both maximum likelihood and least square estimators benefit from the Markov property for quicker computations. Indeed, their complexity is linear in the total number of events N_T instead of being quadratic as in the generic case. While less commonly used, the least square estimator is a quadratic function which is very easy to optimize unlike the maximum likelihood estimator. Also, it scales very well with the total number of events since a level of precision as tight as wanted can be reached with $D^2 \times U^2$ passes on the data, see Chapter I for more details.

We will now focus on social network information propagation that is a challenging problem of fastly growing interest [dMB04, Les08, CS08, LBK09] because of the large number of applications in web-advertisement and e-commerce, where large-scale logs of event history are available. A common supervised approach consists in the prediction of labels based on declared interactions (friendship, like, follower, etc.). However, such supervision is not always available, and it does not always describe accurately the level of interactions between users. Labels are often only binary while a quantification of the interaction is more interesting, declared interactions are often deprecated, and more generally a supervised approach is not enough to infer the latent communities of users, as temporal patterns of actions of users are much more informative. With Hawkes processes we consider an approach directly built on the unaltered data of users actions. Formally, the users are the nodes of the multivariate point process and their timed actions are the events. Though, a raw Hawkes process would hardly recover the characteristic patterns usually observed on a social network such as the sparse interactions among users and the community structures. This leads to following interrogation,

Question 1. *How to retrieve social interaction dynamics with Hawkes processes ?*

We discuss this problematic in the following section and present a numerically efficient technique inspired by a strong theoretic analysis.

2 Sparse and low-rank multivariate Hawkes processes

We consider the problem of unveiling the implicit network structure of user interactions in a social network, based only on high-frequency timestamps. Recently, an approach built on Hawkes processes has increased in popularity [CS08, BBH12, ZZS13a, YZ13]. It uses Hawkes structure to retrieve the direct influence of a specific user's action on all the future actions of all the users (including himself). Hawkes processes model simultaneously decay of the influence over time with the shape of the kernels φ^{ij}, the levels of interaction between nodes with the weighted asymmetrical adjacency matrix Φ and the baseline intensity, that measures the level of exogeneity of a user, namely the spontaneous apparition of an action, with no influence from other nodes of the network.

2.1 ℓ_1 and trace-norm penalizations

Penalization (also called regularization) is a very common technique in machine learning. It consists in minimizing the opposite goodness-of-fit function plus a penalty term which can be designed to enforce a specific structure on the problem minimizer. Lasso or ℓ_1 penalty [Tib96] is one of the best known and most used. In a problem where $F : \mathbb{R}^d \to \mathbb{R}$ is the opposite goodness-of-fit function, its ℓ_1-penalized version consists in solving

$$\min_{w \in \mathbb{R}^d} f(w) + \lambda \|w\|_1,$$

where $\lambda > 0$ and $\|w\|_1 = \sum_{j=1}^d |w_j|$. This penalty is known to output solutions with sparse support, that is a vector of coefficients w with many zeros entries. In fact, if we consider a more general penalty of the form of ℓ_p penalization, $\sum_{j=1}^d |w_j|^p$, then the lasso is the ℓ_1 penalization and the ℓ_0 penalization, which gives the number of nonzero entries of a vector, is the limiting case when $p \to 0$. In this setting, lasso uses the smallest value of p that leads to a convex formulation of an ℓ_p-penalized problem. Hence, lasso can also be viewed as a convex relaxation of the best subset selection problem [Tib11].

The *trace-norm* penalization (also known as nuclear norm) is used when the coefficients $\Omega \in \mathbb{R}^{d \times d}$ are matrices instead of vectors. It consists in adding the trace-norm of the matrix $\|\Omega\|_*$ to the opposite goodness-of-fit, given by

$$\|\Omega\|_* = \sum_{j=1}^{d} \sigma_j(\Omega),$$

where $\sigma_1(\Omega) \geq \cdots \geq \sigma_d(\Omega) \geq 0$ are the singular values of Ω. Thus, this is equivalent to an ℓ_1-penalization of the vector $[\sigma_1(\Omega) \ \cdots \ \sigma_d(\Omega)]$ and tends to enforce the ap-

parition of zero singular values. Finally, as the rank of matrix equals the number of non-zero singular values, trace-norm penalization is a convex relaxation of the low-rank problem [CT10], just like ℓ_1 penalization is for the best subset selection problem. Enforcing low-rank is now of common use in in collaborative filtering problems [CT04, CT10, RRS11] to describe the network structure with a limited number of parameters. Popularized by the Netflix prize [BL07], this tends to make appear communities of individuals with similar behaviors.

2.2 Sparse and low rank priors

We apply these penalizations technique on a modeling with Hawkes process of d nodes, each of them representing a user. For more simplicity, we consider in this chapter that, for all $j, j' = 1, \dots, d$ the kernels $\varphi_{j,j'}$ can be decomposed into $\varphi_{j,j'}(t) = \sum_{k=1}^{K} a_{j,j',k} h_{j,j',k}(t)$ where $a_{j,j',k} \geq 0$ and $h_{j,j',k} : \mathbb{R}^+ \to \mathbb{R}^+$ are known decay functions with fixed ℓ_1 norm, $\|h_{j,j',k}\|_1 = 1$. Hence, for each node $j = 1, \dots, d$, the conditional intensity writes

$$\lambda_{j,\mu,\mathbb{A}}(t) = \mu_j + \sum_{j'=1}^{d} \sum_{k=1}^{K} a_{j,j',k} h_{j,j',k}(t-s) \, \mathrm{d}N_s^j,$$

where $\mu \in \mathbb{R}^d$ is the exogeneous intensity. Typically, choosing $h_{j,j',k} = \beta_k \exp(-\beta_k t)$ where $\beta_k > 0$, reverts to the sum of exponentials kernel parametrization (see Section 1.2). The parameter of interest is the *self-excitement* tensor $\mathbb{A}$ that simply rewrites here $\mathbb{A} = [a_{j,j',k}]_{1 \leq j,j' \leq d, 1 \leq k \leq K}$. Then, we combine convex proxies for sparsity and low-rank of self-excitement tensor and the baseline intensities to obtain the desired network structure. Our prior assumptions on μ and $\mathbb{A}$ are the following.

Sparsity of μ. Some nodes are basically inactive and react only if stimulated. Hence, we assume that the baseline vector μ is sparse.

Sparsity of $\mathbb{A}$. A node interacts only with a fraction of other nodes, meaning that for a fixed node j, only a few $a_{j,j',k}$ are non-zero. Moreover, a node might react at specific time scales only, namely $a_{j,j',k}$ is non-zero for some k only for fixed j, j'. Thus, we assume that $\mathbb{A}$ is an entrywise sparse tensor.

Low-rank of $\mathbb{A}$. We assume that there exist latent factors that explain the way nodes impact other nodes through the different scales $k = 1, \dots, K$. Rewriting the intensity naturally leads to penalize the rank of $d \times Kd$ matrix $\mathrm{hstack}(\mathbb{A}) = \left[\mathbb{A}_{\bullet,\bullet,1} \cdots \mathbb{A}_{\bullet,\bullet,K}\right]$ where $\mathbb{A}_{\bullet,\bullet,k}$ stands for the $d \times d$ matrix with entries $(\mathbb{A}_{\bullet,\bullet,k})_{j,j'} = \mathbb{A}_{j,j',k}$.

We induce these proxies with penalty terms added to the objective. Namely, we minimize

$$R(\mu, \boldsymbol{A}) + \|\mu\|_{1,\hat{w}} + \|\mathbb{A}\|_{1,\hat{\mathbb{W}}} + \hat{\tau} \|\mathrm{hstack}(\mathbb{A})\|_*, \tag{2}$$

where $R(\mu, \boldsymbol{A})$ is the least squares goodness-of-fit (1), and the penalty terms are trace-norm and weighted ℓ_1 penalizations, given by

$$\|\mu\|_{1,\hat{w}} = \sum_{j=1}^{d} \hat{w}_j |\mu_j|, \quad \|\mathbb{A}\|_{1,\hat{\mathbb{W}}} = \sum_{1 \le j,j' \le d, 1 \le k \le K} \hat{\mathbb{W}}_{j,j',k} |\mathbb{A}_{j,j',k}|.$$

The weights $\hat{w}$, $\hat{\mathbb{W}}$, and the coefficient $\hat{\tau}$ are data-driven tuning parameters given in Section 4 of Chapter II. The choice of these weights lead to a data-driven scaling of the variability of information available for each nodes and comes from a sharp analysis of the noise terms presented in the section below.

2.3 Sharp oracle inequality

With a wise choice of the weights parameters $\hat{w}$, $\hat{\mathbb{W}}$ and $\hat{\tau}$, we derive, in Theorem 1, an oracle inequality. This inequality bounds the estimation error of the intensity $\lambda_{\hat{\mu},\hat{\mathbb{A}}}$, obtained by minimizing Problem (2), given the best possible estimator, with the same parametrization, that would rely on perfect information. We fix some confidence level $x > 0$, which can be safely chosen as $x = \log T$ for instance, and denote by $\|\cdot\|_F$ the Frobenius norm to formulate the following theorem where no assumption is made on the ground truth intensity λ.

Theorem 1. *Fix $x > 0$, and let $\hat{w}, \hat{\boldsymbol{W}}, \hat{\tau}$ that depends on x be given by* (II.17), (II.18) *and* (II.19). *Then, the inequality*

$$\|\lambda_{\hat{\mu},\hat{\mathbb{A}}} - \lambda\|_T^2 \le \inf_{\mu,\mathbb{A}} \Big\{ \|\lambda_{\mu,\mathbb{A}} - \lambda\|_T^2 + 1.25\kappa(\mu,\mathbb{A})^2 \Big(\|(\hat{w})_{\mathrm{supp}(\mu)}\|_2^2 + \|(\hat{\mathbb{W}})_{\mathrm{supp}(\mathbb{A})}\|_F^2 + \hat{\tau}^2 \,\mathrm{rank}(\mathrm{hstack}(\mathbb{A})) \Big) \Big\}$$

holds with a probability larger than $1 - 70.35e^{-x}$.

The constant $\kappa(\mu,\mathbb{A})$ is given by Definition 1 of Chapter II. It comes from the necessity to have a restricted eigenvalue condition on the Gram matrix of the problem to obtain an oracle inequality with a fast rate [BRT09, Koll1]. Roughly, it requires that for any set of parameters $\{\mu', \mathbb{A}'\}$ that has a support close to the one of $\{\mu, \mathbb{A}\}$, we have that the L^2 norm of $\{\mu', \mathbb{A}'\}$ in the support of $\{\mu, \mathbb{A}\}$ can be bounded by the L^2 norm of the intensity given by $\|\lambda_{\mu',\mathbb{A}'}\|_T$.

2.4 Numerical experiments

To measure the performances of the choice of the data-driven weighting of the penalizations $\{\hat{w}, \hat{\mathbb{W}}, \hat{\tau}\}$, we conduct experiments on synthetic datasets and compare our

method to non-weighted penalizations [ZZS13a]. We perform these experiments on Hawkes processes with $d = 30$ nodes and $K = 3$ basis kernels where the self-excitement tensor contains square overlapping boxes (corresponding to overlapping communities) to respect the sparse and low rank priors. We consider four estimation procedures that minimize the least square goodness-of-fit plus one of the following penalization:

- L1: non-weighted ℓ_1 penalization of $\mathbb{A}$
- wL1: weighted ℓ_1 penalization of $\mathbb{A}$
- L1Nuclear: non-weighted ℓ_1 penalization and trace-norm penalization of hstack($\mathbb{A}$) (same as [ZZS13a])
- wL1Nuclear: weighted ℓ_1 penalization and trace-norm penalization of hstack($\mathbb{A}$)

Then, for each procedure, we train the model on the generated data, restricting it on a growing time intervals, and assessing their performance each time with the three following metrics:

- Estimation error: the relative ℓ_2 estimation error of $\mathbb{A}$, given by $\|\hat{\mathbb{A}} - \mathbb{A}\|_2^2 / \|\mathbb{A}\|_2^2$
- AUC: we compute the AUC (area under the ROC curve) between the binarized ground truth matrix $\mathbb{A}$ and the solution $\hat{\mathbb{A}}$ with entries scaled in $[0, 1]$. This allows us to quantify the ability of the procedure to detect the support of the connectivity structure between nodes.
- Kendall: we compute Kendall's tau-b between all entries of the ground truth matrix $\mathbb{A}$ and the solution $\hat{\mathbb{A}}$. This correlation coefficient takes value between -1 and 1 and compare the number of concordant and discordant pairs. This allows us to quantify the ability of the procedure to rank correctly the intensity of the connectivity between nodes.

Figure 2 confirms that weighted penalizations systematically leads to an improvement, both for L1 and L1Nuclear, in terms of error, AUC and Kendall coefficient.

Studying the optimization techniques used to minimize the objective (2) was beyond the scope of this section. However, optimization is a crucial part of the procedure on which we will focus in the two following sections.

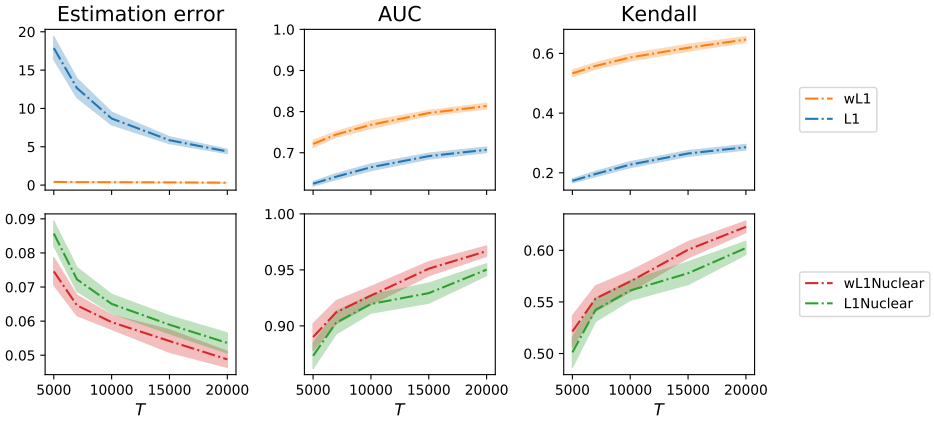

Figure 2: Metrics values for simulated data of dimension $d = 30$ and $K = 3$ basis kernels. Abscissa corresponds to the interval length T. Weighted penalizations systematically leads to an improvement, both for L1 and L1 + Nuclear penalization.

3 Background on composite sum minimization with first order methods

A wide variety of machine learning tasks consists in optimizing the following problem

$$\min_{w \in \mathbb{R}^d} F(w) \quad \text{with} \quad F(w) = f(w) + g(w), \quad f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w),$$

where the function f_i corresponds to a loss computed at a sample i of the dataset and the convex function g is a penalization term. This framework includes classification with logistic regression with $f_i(w) = \log(1 + \exp(-y_i w^\top x_i))$, least square regression with $f_i(w) = (y_i - w^\top x_i)^2$ among many others. It is common to assume that function f is gradient-Lipschitz, namely $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for any $x, y \in \mathbb{R}^d$ where $\|.\|$ stands for the Euclidean norm on $\mathbb{R}^d$, and $L > 0$ is the Lipschitz constant. With this property, the *descent lemma* [Ber99, Proposition A.24] holds,

$$f(w + \Delta w) \leq f(w) + \Delta w^\top \nabla f(w) + \tfrac{L}{2}\|\Delta w\|^2$$

for any $w, \Delta w \in \mathbb{R}^d$. Most first order algorithms ensue from this lemma and firstly the gradient descent algorithm where at each step Δw^{t+1} is set to the optimal value $-\frac{1}{L}\nabla f(w^t)$. The penalization term is then handled with proximal operators [CP11]. This leads to ISTA algorithm and its accelerated version FISTA [BT09] whose rate is optimal [Nes83].

Stochastic gradient descent. However, with its structure, this problem can also be considered as an accumulation of smaller problems f_i that have a common behavior. Stochastic gradient descent (SGD) [RM51] exploits this and instead of computing the full gradient $\nabla f(w^t)$ at each step, uses a random variable $\phi_t \in \mathbb{R}^d$ such that $\mathbb{E}[\phi_t] = \nabla f(w^t)$. The descent iteration becomes $\Delta w^{t+1} = -\eta_t \phi_t$ where ϕ_t is set to $\nabla f_i(w^t)$ and $\eta_t > 0$ is a step size. When all $\nabla f_i(w^t)$ are as expensive to compute, SGD performs iterations that are n times quicker than the batch algorithms previously introduced. But, this method does not converge easily to a precise solution because $\nabla f_i(w^t)$ does not approach zero when w^t is close from the optimal value w^*. Hence, the sequence of step size $(\eta_t)_{t\geq 0}$ must be decreasing which eventually affects the convergence speed.

Variance reduced stochastic algorithms. Recently, stochastic solvers based on a combination of SGD and the Monte-Carlo technique of variance reduction [SLRB17], [SSZ13], [JZ13], [DBLJ14] turn out to be both very efficient numerically (each update has a complexity comparable to vanilla SGD) and very sound theoretically. To reduce its variance, the random variable ϕ_t is set to $\nabla f_i(w^t) + Y - \mathbb{E}[Y]$ where $Y \in \mathbb{R}^d$ is a random variable that is expected to be correlated with $\nabla f_i(w^t)$. Hence, ϕ_t remains an unbiased estimator of $\nabla f(w^t)$ and its variance is decreased. In [SLRB17], [SSZ13], [JZ13] and [DBLJ14], ϕ^t converges to 0 at the optimum, and the sequence of step sizes $(\eta_t)_{t\geq 0}$ does not need to be decreasing as in SGD. Theses algorithms obtain linear convergence rate, that is they reach an iterate w^t such that $F(w^t) \leq F(w^*) + \varepsilon$ in less than $\mathcal{O}(\log(1/\varepsilon))$ iterations.

Thus, modern optimization methods obtain high precision solutions with few passes on the data. However, the first order algorithms that we have just introduced rely on the assumption that f is gradient-Lipschitz which is not verified by Hawkes processes log likelihood. The lack of fast, scalable and robust method to solve this problem motivates the following question.

Question 2. *How to optimize non gradient-Lipschitz objectives such as Hawkes processes log-likelihood ?*

We focus on this particular problem in the next section where we develop an algorithm dedicated to a new class of functions, admitting another smoothness assumption.

4 Dual optimization for convex constrained objectives without the gradient-Lipschitz assumption

When the gradient-Lipschitz assumption is not verified, descent lemma does not hold anymore and the previous algorithms have no more convergence guarantees. Motivated by learning problems that do not meet this assumption, such as linear Poisson regression and Hawkes processes, we work under another smoothness assumption, and obtain a linear convergence rate for a shifted version of SDCA (Stochastic Dual Coordinate Ascent) [SSZ13] that improves the current state-of-the-art.

SDCA for log **smooth objectives**. In order to remove the gradient-Lipschitz assumption, we need to focus on a more specific task relying on a new smoothness assumption. Given convex functions $f_i : \mathscr{D}_f \to \mathbb{R}$ with $\mathscr{D}_f = (0, +\infty)$ such that $\lim_{t\to 0} f_i(t) = +\infty$, $\psi \in \mathbb{R}^d$, $x_1, \dots, x_n \in \mathbb{R}^d$, $\lambda > 0$ and given a 1-strongly convex function $g : \mathbb{R}^d \to \mathbb{R}$, we consider the objective

$$\min_{w \in \Pi(X)} P(w) \quad \text{where} \quad P(w) = \psi^\top w + \frac{1}{n}\sum_{i=1}^{n} f_i(w^\top x_i) + \lambda g(w), \tag{3}$$

where $\Pi(X)$ is the polytope $\{w \in \mathbb{R}^d \,:\, \forall i \in \{1, \dots, n\},\ w^\top x_i > 0\}$ that we assume to be non-empty. The previously introduced first order algorithms have no theoretical guarantees for this problem and are unable to maintain their iterates w^t in the polytope $\Pi(X)$ in our experiments. To deal with simpler constraints we rather focus on the dual problem which is only box-constrained,

$$\max_{\alpha \in -\mathscr{D}_{f^*}^n} D(\alpha) \quad \text{where} \quad D(\alpha) = \frac{1}{n}\sum_{i=1}^{n} -f_i^*(-\alpha_i) - \lambda g^*\left(\frac{1}{\lambda n}\sum_{i=1}^{n} \alpha_i x_i - \frac{1}{\lambda}\psi\right),$$

where f^* (resp. g^*) is the Fenchel conjugate of f (resp. g) and $-\mathscr{D}_{f^*}^n$ is the domain of the function $x \mapsto \sum_{i=1}^n f_i^*(-x)$. While, it is not straightforward to obtain strong duality for a convex problem with open constraints, it is guaranteed in such a setting. (see Proposition 1 of Chapter IV). Hence, we maximize this dual with a shifted variant of SDCA algorithm [SSZ13] that does not rely on gradient-Lipschitz assumption but rather on the following log smoothness property.

Definition 1. *We say that a function $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ is L-*log smooth*, where $L > 0$, if it is a differentiable and strictly monotone convex function that satisfies*

$$|f'(x) - f'(y)| \leq \frac{1}{L} f'(x) f'(y) |x - y|$$

for $\forall x, y \in \mathscr{D}_f$.

In fact, in view of the following proposition, log smoothness is linked to the self-concordance property introduced by Nesterov [Nes13] and widely used to study losses involving logarithms.

Proposition 1. *Let $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ be a convex strictly monotone and twice differentiable function. Then,*

$$f \text{ is } L\text{-log } \textit{smooth} \quad \Leftrightarrow \quad \forall x \in \mathscr{D}_f,\ f''(x) \leq \tfrac{1}{L} f'(x)^2.$$

It appears that log smoothness is the counterpart of self-concordance but to control the second order derivative with the first order derivative. Assuming that all functions f_i are L_i log smooth, we derive new tight convex inequalities and prove in Chapter IV the following theorem of convergence where $\alpha^* \in \mathbb{R}^n$ is the maximizer of the dual objective.

Theorem 2. *Suppose that we known bounds $\beta_i \in -\mathscr{D}^n_{f^*}$ such that $R_i = \frac{\beta_i}{\alpha_i^*} \geq 1$ for $i = 1, \ldots, n$ and assume that all f_i are L_i-log smooth and g is 1-strongly convex. Then SDCA satisfies*

$$\mathbb{E}[D(\alpha^{(t)}) - D(\alpha^*)] \geq \left(1 - \frac{\min_i \sigma_i}{n}\right)^t \left(D(\alpha^*) - D(\alpha^{(0)}\right),$$

where

$$\sigma_i = \left(1 + \frac{\|x_i\|^2 \alpha_i^{*2}}{2\lambda n L_i} \frac{(R_i - 1)^2}{\frac{1}{R_i} + \log R_i - 1}\right)^{-1}.$$

This theorem gives a linear convergence rate for the dual objective that improves what is obtained with regular SDCA analysis. We then improve these theoretical guarantees by providing an importance sampling variant of our algorithm, and its numerical efficiency with a heuristic initialization and a mini-batch method.

Application to Poisson regression and Hawkes processes Linear Poisson regression is widely used in image reconstruction [HMW12], web-marketing [CPC09] and survival analysis to model additive effects, as opposed to multiplicative effects [BF10]. SDCA for log smooth objectives applies to linear Poisson regression and also to the previously introduced Hawkes processes with sum of exponentials kernels. For both models, we can formulate their likelihood as in Equation (3) and give explicit candidates for the bounds β_i required by Theorem 2. While the problem formulation is straightforward for Poisson regression, it involves precomputed weights for Hawkes processes and leads to I independent subproblems where I is the number of nodes of the Hawkes process.

Experimentally, we compare SDCA with a second order algorithm that is the standard second-order Newton algorithm which computes at each iteration the hessian of the objective which is then used to solve a linear system. This ensures supra-linear

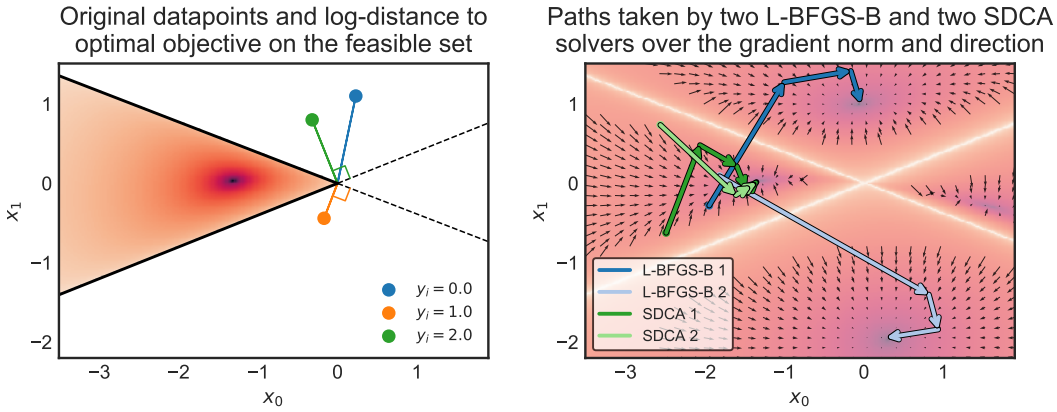

Figure 3: Iterates of SDCA and L-BFGS-B on a Poisson regression toy example with three samples and two features. *Left.* Dataset and value of the objective. *Right.* Iterates of L-BFGS-B and SDCA with two different starting points. The background represents the gradient norm and the arrows the gradient direction. SDCA is very stable and converges quickly towards the optimum, while L-BFGS-B easily converges out of the feasible space.

convergence guarantees and keeps all iterates in the open polytope $\Pi(X)$ as soon as the starting point is in it [NN94]. However, this algorithm scales very poorly with the number of dimensions d (the size of the vector w) limiting drastically its usage in practice. Next, we compare SDCA with SVRG [JZ13, TMDQ16] and the limited-memory quasi-Newton L-BFGS-B algorithm [Noc80, NW06]. They both theoretically rely on the gradient-Lipschitz smoothness assumption which is not verified here. In practice, they very often diverge and violate of the open polytope constraint $\Pi(X)$. This is illustrated in the toy example of Figure 3. Hence, in order to obtain comparable results, we manage to force the constraint by the projecting the iterates of SVRG and L-BFGS-B onto $[0, +\infty)^d$.

As expected, in Figure 4, we observe that the Newton algorithm becomes very slow when the number of features d increases and that SVRG and L-BFGS-B cannot reach the optimal solution because their iterates are constrained to $[0, +\infty)^d$ while the problem minimizer contains negative values. SDCA is the only first order solver that can reach the optimal solution and combines the best of both worlds, the scalability of a first order solver and the ability to reach solutions with negative entries.

Hawkes processes and convex optimization of finite sum objectives are two fields of growing interest. In both cases, the numerical results are of prime importance but articles are rarely published with code that not only reproduces the experiments but

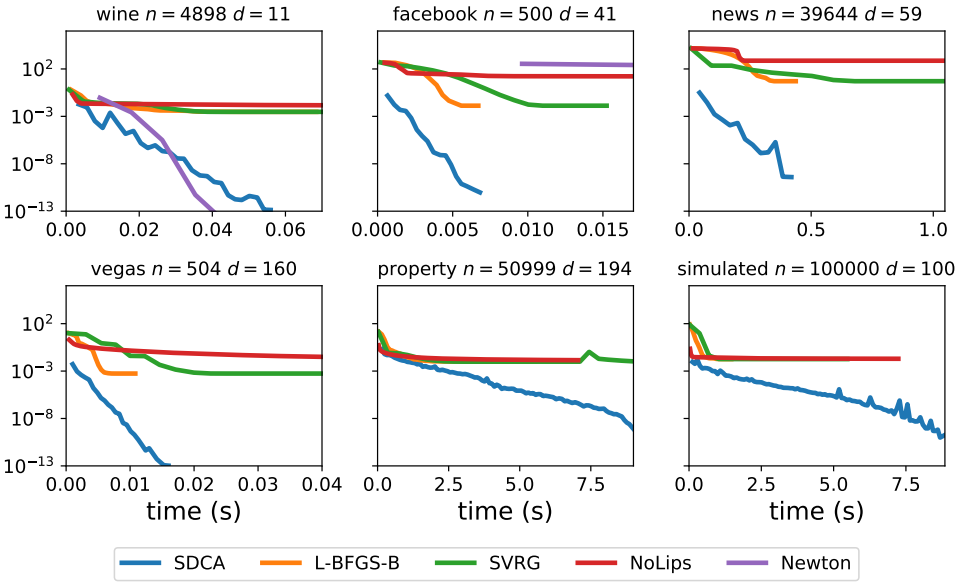

Figure 4: Convergence over time of four algorithms SDCA, SVRG, L-BFGS-B and Newton on 6 datasets of Poisson regression. SDCA combines the best of both worlds: speed and scalability of SVRG and L-BFGS-B with the precision of Newton's solution.

is also meant to be reused for further applications. This makes convex optimization solvers or Hawkes processes estimators difficult to compare in a unified manner and raises the following question.

Question 3. *How to make these statistical inference tools available to a wide audience ?*

In the following section, we present a new Python library addressing both convex optimization and Hawkes processes to facilitate their practical usage.

5 tick: a Python library for statistical learning

`tick` is a statistical learning library for Python 3, with a particular emphasis on time-dependent models, such as point processes, tools for generalized linear models and survival analysis. It relies on a C++ implementation and state-of-the-art optimization algorithms to provide very fast computations in a single node multi-core setting. Source code and documentation can be downloaded from `https://github.com/X-DataInitiative/tick`.

Table 1: Models and estimation techniques for Hawkes processes available in `tick`

Non Parametric	Parametric
EM [LM11]	Single exponential kernel
Basis kernels [ZZS13a]	Sum of exponentials kernels
Wiener-Hopf [BM14]	Sum of gaussians kernels [XFZ16]
NPHC [ABG+17]	ADM4 [ZZS13a]

Hawkes Despite the growing interest in Hawkes models, very few open source packages are available. There are mainly three of them. The library `pyhawkes`[1] proposes a small set of Bayesian inference algorithms for Hawkes process. `hawkes` R[2] is an R-based library that provides a single estimation algorithm, and is hardly optimized. Finally, `PtPack`[3] is a C++ library which proposes parametric maximum likelihood estimators, with sparsity-inducing regularizations. Written in Python, `tick` is the most comprehensive library that deals with Hawkes processes for instance, by including the main inference algorithms from the literature listed in Table 1. This encompasses both parametric and non parametric algorithms and brings them to a new accessibility level.

Toolbox for convex optimization Besides Hawkes processes, `tick` has three main modules: `tick.linear_model` with linear, logistic and Poisson regression, `tick.robust` for robust linear models and `tick.survival` for survival analysis. At a high level `tick` follows `scikit-learn`'s API [PVG+11, BLB+13] which is well-known for its completeness and ease of use but under the hood, these modules rely on a convex optimization toolbox built to solve composite sum minimization Problem (3). This toolbox allows to combine with many models different penalization techniques (`tick.prox` module) and state-of-the-art optimization algorithms (`tick.solver`). It is implemented in a very modular way and allows more possibilities than other `scikit-learn` API based optimization libraries such as `lightning`[4]. A non exhaustive list of possible combinations is given in Table 2 and highlights how useful `tick` is to run experiments such as testing a new model with various penalization techniques or comparing convex optimization solvers.

Implementation While `tick` is a Python library, all the heavy computations run in C++ which communicates with Python with SWIG (Simplified Wrapper and Interface

[1] https://github.com/slinderman/pyhawkes
[2] https://cran.r-project.org/web/packages/hawkes/hawkes.pdf
[3] https://github.com/dunan/MultiVariatePointProcess
[4] http://contrib.scikit-learn.org/lightning

Table 2: `tick` allows the user to combine many models, prox and solvers. This list is not exhaustive.

Model	Proximal operator	Solver
Linear regression	SLOPE	Gradient Descent
Logistic regression	L1 (Lasso)	Stochastic Variance Reduced Gradient
Poisson regression	Total Variation	Stochastic Gradient Descent
Cox regression	Group L1	Accelerated Gradient Descent
Hawkes with exp. kernels	L2 (Ridge)	Stochastic Dual Coordinate Ascent

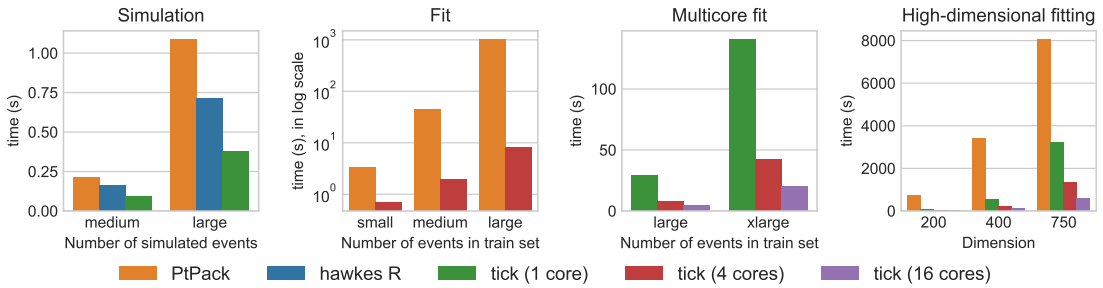

Figure 5: Computational timings of `tick` versus `PtPack` and `hawkes` R. `tick` strongly outperforms both libraries for simulation and fitting (note that the "Fit" graph is in log-scale). The model fits in plots "Fit" and "Multicore fit" are compared on simulated 16-dimensional Hawkes processes, with an increasing number of events: small=5×10^4, medium=2×10^5, large=10^6, xlarge= 5×10^7, while 200, 400 and 750 dimensional Hawkes processes are fitted in plot "High-dimensional fitting". "Multi-core fit" and "High-dimensional fitting" plots show that `tick` benefits from multi-core environments to speed up computations.

Generator) [BFK+96]. Thanks to SWIG, the Python objects have a very easy access to full C++ objects that share their memory and work on the same dataset without needing any copy. This is particularly useful for optimization toolbox where `model`, `prox` and `solver` are symbolically linked in Python and then run fully in C++. Also, the C++ part of the library is independent and with some effort is usable without the Python part. This allows the developers to analyze the code with any profiling tool compatible with C++ and hence produce code optimized in depth. For all these reasons, `tick` is a very fast library and has proved to be up to an order of magnitude faster than `hawkes` R and `PtPack` on a series of benchmarks presented in Figure 5.

CHAPTER I

Background on Hawkes processes

1 Temporal point processes

Temporal datasets are generally explored with *time series* in very various fields such as finance [Tsa05, Tay08], weather forecasting [Bur03], econometrics [LK04] or astronomy [Sca82]. Time series work with measures taken at regular intervals on a discrete time scale. The length of this interval, that is the time resolution, is a sensitive parameter. For example, it must be adapted depending on if you want to study short-term or long-term interactions. On the contrary, *temporal point processes* use sequences of events in continuous time and can study multiple time scales at once. This chapter gives a short introduction on temporal point processes with a specific focus on Hawkes processes, further details can be found in [DVJ07].

1.1 Definition

We associate to a set of distinct timestamps $\{t_1, \dots, t_n\}$ occurring in a time interval $[0, T]$, the counting process $N_t = \sum_{t_k} \mathbb{1}_{t_k \leq t}$. This counting process is a random process which evolves over time by jumps of size 1. Studying temporal point processes consists in analyzing when this jumps occur. This behavior is characterized by an *intensity function* $\lambda(t|\mathcal{F}_t)$ which gives the infinitesimal probability with which an event will arise at time t given the information $\mathcal{F}_t$ available up to (but not including) time t. It writes

$$\lambda(t|\mathcal{F}_t) = \lim_{dt \to 0} \frac{\mathbb{P}(N_{t+dt} - N_t = 1 |\ \mathcal{F}_t)}{dt}.$$

This intensity fully characterizes a point process. In the most simple case, this intensity is constant and the associated process is called a homogeneous Poisson process. It describes a phenomenon with no memory and a constant probability of occurrence in which $N_{t+\Delta t} - N_t$ follows a Poisson distribution of parameter Δt for any $\Delta t > 0$.

More generally, a process whose intensity is not constant but depends on time t only, is called an inhomogeneous Poisson process.

1.2 Goodness-of-fit

We call goodness-of-fit a function telling how well a statistical model fits a set of observations. In our context, the most common goodness-of-fit measure is the likelihood of point processes given by [DVJ07]. For convenience we rather consider the opposite of its logarithm as an error measure, given by

$$-\log L(\lambda, \mathscr{F}_T) = \int_0^T \lambda(s|\mathscr{F}_s)\,\mathrm{d}s - \sum_{k=1}^{N_T} \log \lambda(t_k|\mathscr{F}_{t_k}), \tag{I.1}$$

where $\mathscr{F}_T = \{t_1, \dots, t_n\}$ is the full history of the process and N_T is the total number of events that have occured in $[0, T]$. Under some assumptions, the maximum likelihood estimator obtained by minimizing this error is consistent, asymptotically normal and asymptotically efficient, see [Oga78] for more details. Another measure is the least squares error inspired by the empirical risk minimization principle. For point processes it writes (see [RBR10, HRBR15])

$$R(\lambda, \mathscr{F}_T) = \int_0^T \lambda(s|\mathscr{F}_s)^2\,\mathrm{d}s - 2\sum_{k=1}^{N_T} \lambda(t_k|\mathscr{F}_{t_k}). \tag{I.2}$$

This writing is quite natural: assuming that the process N_t has an unknown ground truth intensity λ^*, then

$$\mathbb{E}\sum_{k=1}^{N_T} \lambda(t_k|\mathscr{F}_{t_k}) = \mathbb{E}\int_0^T \lambda(s|\mathscr{F}_s)\,\mathrm{d}N_s = \int_0^T \lambda(s|\mathscr{F}_s)\lambda^*(s)\,\mathrm{d}s,$$

and the expectation of $R(\mathscr{F}_T)$ rewrites

$$\mathbb{E}[R(\lambda, \mathscr{F}_T)] = \mathbb{E}\|\lambda\|_T^2 - 2\mathbb{E}\langle\lambda|\ \lambda^*\rangle_T = \mathbb{E}\|\lambda - \lambda^*\|_T^2 - \|\lambda^*\|_T^2,$$

where $\langle\lambda|\ \lambda^*\rangle_T = \int_0^T \lambda(s|\mathscr{F}_s)\lambda^*(s|\mathscr{F}_s)\,\mathrm{d}s$ and $\|\lambda\|_T^2 = \langle\lambda|\ \lambda\rangle_T$. Hence, $\mathbb{E}[R(\lambda, \mathscr{F}_T)]$ is minimized by λ^*. This second goodness-of-fit is not as common but is easier to optimize in some cases such as Hawkes processes parametrized with exponential kernels (see Section 2.4). In both cases, these measures encourage intensity functions with high values at times when events occur and low at any other time.

2 Hawkes processes

Hawkes processes [Haw71a, HO74] are temporal point processes in which the intensity depends on the process history with an excitation mechanism. They can

be understood as the equivalent of autoregressive time series models (AR) [Ham94] but in continuous time. This allows to study cross causality that might occur in one or several events series. They have first been used to study earthquake propagation [Oga99], the network across which the aftershocks propagate can be recovered given all tremors timestamps. Then, they have been applied to high frequency finance [BDHM13, BMM15] to describe market reactions to different types of orders. In the recent years Hawkes processes have found many new applications including crime prediction [LMBB12, Moh13], social network information propagation [CS08, BBH12, ZZS13a, YZ13, LSV+16] or neuron spike modeling [GL15, HRBR15].

2.1 Multivariate Hawkes processes

Multivariate Hawkes processes model the interactions of $D \geq 1$ temporal point processes. Namely, it models timestamps $\{t_k^i\}_{k\geq 1}$ of nodes $i = 1,\dots,D$ associated with a multivariate counting process $N_t = [N_t^1 \cdots N_t^D]$. The excitation dynamic between the nodes is encompassed by the auto-regressive structure of the conditional intensity. For component N_t^i it writes

$$\lambda^i(t|\,\mathscr{F}_t) = \mu^i + \sum_{j=1}^{D} \int_0^t \varphi^{ij}(t-s)\,\mathrm{d}N_s^j. \tag{I.3}$$

The $\mu^i \geq 0$ are called *baseline* intensities and correspond to the exogenous intensities of events for node $i = 1,\dots,D$. The φ^{ij} for $1 \leq i,j \leq D$ are called *kernels*, they quantify over time and in magnitude the influence of past events from node j on the intensity of events from node i. These kernel functions are positive and causal (their support is within $\mathbb{R}^+$) and if they are integrable each entry of the $D \times D$ integral matrix $(\Phi)_{i,j} = \int_0^T \varphi^{ij}(t)\,\mathrm{d}t$ denotes the expected number of events of type i directly triggered by an event of type j.

This observation leads to the *population representation* [HO74] of the Hawkes processes in which we consider D populations whose count increases in two manners, either with new migrants or with children from the existing individuals. In this representation, we consider that the arrivals of the migrants of population $i = 1,\dots,D$ are modeled with a homogeneous Poisson process of intensity μ^i and each individual of any population $j = 1,\dots,D$ arrived or born at time t gives birth to children of population i following an inhomogeneous Poisson process of intensity $\varphi^{ij}(\,\cdot\, - t)$.

This representation makes clear the necessity of a stability condition to avoid an explosion of the number of individuals and reach a stationary regime. In such a regime, the mean intensity of node i writes

$$\bar{\lambda}^i = \mu^i + \sum_{j=1}^{D} \mathbb{E}\int_{-\infty}^{t} \varphi^{ij}(t-s)\,\mathrm{d}N_s^j = \mu^i + \sum_{j=1}^{D} \bar{\lambda}^j (\Phi)_{i,j},$$

which with $\bar{\lambda} = [\bar{\lambda}^1 \cdots \bar{\lambda}^D]$ and $\mu = [\mu^1 \cdots \mu^D]$ gives,

$$\bar{\lambda} = \mu + \Phi\bar{\lambda} = \mu(1 - \Phi)^{-1}$$

where the second equality is only valid if the spectral radius of the matrix Φ is strictly smaller that 1: $\rho(\Phi) < 1$ [BMM15]. We call this assumption the stability condition.

2.2 Simulation

The thinning algorithm [Oga81] is the most suitable method to simulate Hawkes processes. This algorithm relies on the existence at all time $\tau \in [0, T]$ of an $\mathscr{F}_\tau$ predictable constant upper bound, $\lambda^*(\tau|\mathscr{F}_\tau)$, of the sum of the conditional intensities of all nodes $\sum_{i=1}^D \lambda^i(t|\mathscr{F}_\tau)$ for any time $t \geq \tau$. At each iteration this algorithm generates an event candidate s sampled from a homogeneous Poisson process of constant intensity $\lambda^*(\tau|\mathscr{F}_\tau)$ where τ is the current time. This candidate is either discarded with probability $1 - \sum_{i=1}^D \lambda^i(s|\mathscr{F}_\tau)/\lambda^*(\tau|\mathscr{F}_\tau)$ or assigned to one node $i = 1, \ldots, n$ with probability $\lambda^i(s|\mathscr{F}_\tau)/\lambda^*(\tau|\mathscr{F}_\tau)$. This procedure is detailed in Algorithm I.1. This algorithm is used in the `tick` library (see Chapter V) to simulate point processes. The following code simulates a 2 nodes Hawkes process with explicitly specified kernels.

Algorithm I.1 Ogata thinning algorithm for multivariate point processes

Require: End time T
  $\tau = 0$
  while $\tau < T$ **do**
    Take $\lambda^* \geq \sum_{i=1}^D \lambda^i(t|\mathscr{F}_\tau)$ for any $t \geq \tau$
    Sample s from an exponential distribution of rate λ^*
    Sample u from a uniform distribution over $[0, 1]$
    for $i = 1, \ldots, D$ **do**
      if $u < \sum_{j=1}^i \lambda^j(s|\mathscr{F}_\tau)/\lambda^*$ **then**
        Append s to node i events
        break
      end if
    end for
    $\tau = s$
  end while

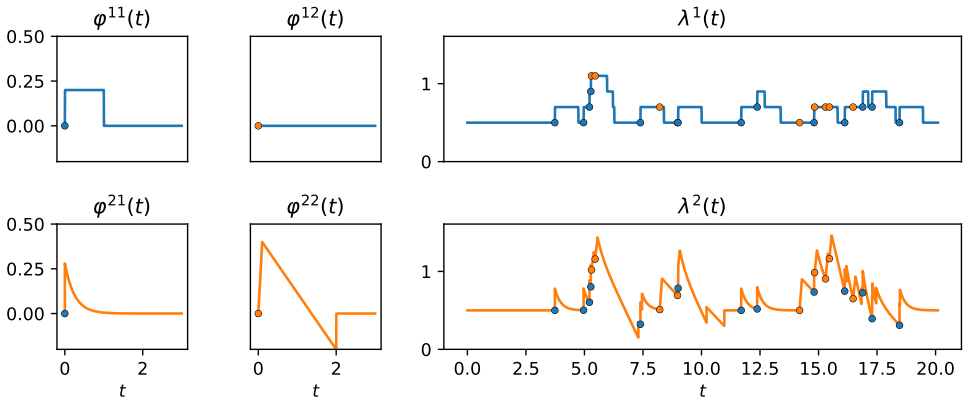

Figure I.1: A realization of a 2 nodes Hawkes process. The four excitation kernels are shown on the left hand side. The intensities are displayed on the right hand side (against time, up to time 20), where events are represented by colored dots (blue corresponding to node 1 and orange to node 2).

```
from tick.base import TimeFunction
from tick.hawkes import (
    SimuHawkes, HawkesKernelExp, HawkesKernelTimeFunc, HawkesKernel0)

tf_11 = TimeFunction([[0, 1], [.2, 0]],
                     inter_mode=TimeFunction.InterConstRight)
kernel_11 = HawkesKernelTimeFunc(tf_11)

kernel_21 = HawkesKernelExp(.07, 4)

tf_22 = TimeFunction([[0, .1, 2], [0, .4, -0.2]])
kernel_22 = HawkesKernelTimeFunc(tf_22)

hawkes = SimuHawkes(kernels=[[kernel_11, 0],
                             [kernel_21, kernel_22]],
                    baseline=[0.5, 0.5], end_time=20)

hawkes.simulate()
```

The realization is shown in Figure I.1. It highlights the excitation mechanism by exhibiting the kernel functions φ^{ij} and the impact of each event on all nodes intensities. For example, one can see that, as encoded in kernel function φ^{12}, the events of type 2 (orange) have no impact on the intensity of node 1 (blue).

2.3 Estimation

Inferring a Hawkes process consists in estimating its exogenous intensity μ and its kernels functions φ^{ij} either in a parametric or in a non-parametric fashion. In the non-parametric case, the kernels are approximated in most cases by histograms [LM11, ZZS13b, BM14]. [LM11] presents the most straight-forward non-parametric algorithm. It assumes that the kernels have a finite support and are piecewise constant on given intervals. Its learning procedure simply consists in Expectation-Maximization steps relying on the population representation [HO74]. This procedure estimates the Hawkes kernels in a very flexible way but is hardly scalable as each of the D^2 kernels is described by many parameters. To lower the number of parameters, [ZZS13b] supposes that all the kernels are linear combinations of basis functions that are learned from the data and hence scales much better with the number of nodes D. Then, [BM14] estimates the kernel functions through the events correlation matrix. In contrary to the first two methods, this one recovers successfully negative kernel functions that encode inhibition effects. Recently, [ABG+17] has provided a new non-parametric estimator that infers directly the matrix Φ of the kernels' norms. The kernel shape is not retrieved anymore but this method scales much better with the number of nodes.

Conversly, parametric estimators rely on a prior knowledge of the kernel functions which are thus described by a set of parameters. Estimating the kernel functions then reverts to estimating these parameters. These procedures are less flexible than non-parametric methods but are more efficient and more robust as the number of parameters to estimate is smaller. They require less events per node and scale better to a higher number of nodes. The chosen parametrization might differ depending on the application and translates desired properties, such as a delayed response to a triggering event [XFZ16] or a slowly decreasing influence with power-law kernels used in seismology [Oga88] and in finance [BMM15]. However, for scalability reason, most parametric estimators are based on exponentially decaying kernels [ELL11, ZZS13a, TFSZ15, FWR+15, LSK17] that are described in the dedicated Section 2.4.

2.4 Exponential kernels

The main parametric model is the so-called *exponential* kernel, in which we consider $\varphi^{ij}(t) = \alpha^{ij}\beta\exp(-\beta t)$ for $\alpha^{ij} > 0$ and $\beta > 0$. In this model the integral matrix $\Phi = [\alpha^{i,j}]_{1\leq i,j\leq d}$ and $\beta > 0$ is a memory parameter. The couple $(N_t, \lambda(t))$ is a Markov process [BMM15, Proposition 2], and Equation (I.3) rewrites in a Markovian form

$$\mathrm{d}\lambda^i(t|\ \mathscr{F}_t) = \sum_{j=1}^{D} \beta(\mu^j - \lambda^j(t|\ \mathscr{F}_t))\,\mathrm{d}t + \alpha^{ij}\beta\,\mathrm{d}N_s^j$$

for $i = 1, \dots D$. This property enables much more efficient computations and a better scaling for both simulation and estimation. However, the maximum likelihood estimator, which is the most common estimation procedure, is efficient only if the decay β is fixed for convexity reasons [LV14]. Hence, the choice of β must be done in advance and the quality of the modeling depends on it.

A more general approach is the *sum of exponentials* kernels [LV14], namely $\varphi^{ij}(t) = \sum_{u=1}^{U} \alpha_u^{ij} \beta_u \exp(-\beta_u t)$ for $\alpha_u^{ij} > 0$ and $\beta_u > 0$. These kernels generalize better as they deal with several time scales β_u which makes the modeling less sensitive to initial choice of the fixed decays. Also, with a carefully chosen list of decays, sum of exponentials kernels can approximate power-law kernels [HBB13, FS15]. Finally, they still benefit from the Markov property if we include the U components of the intensity. With

$$\lambda^i(t|\,\mathscr{F}_t) = \mu^i + \sum_{u=1}^{U} \nu_u^i(t|\,\mathscr{F}_t) \quad \text{where} \quad \nu_u^i(t|\,\mathscr{F}_t) = \sum_{j=1}^{D} \int_0^t \alpha_u^{ij} \beta_u \exp(-\beta_u (t-s))\, \mathrm{d}N_s^j,$$

we have that $(N_t, \nu(t))$ is a Markov process with the following relation

$$\mathrm{d}\nu_u^i(t|\,\mathscr{F}_t) = \sum_{j=1}^{D} -\beta_u \nu_u^j(t|\,\mathscr{F}_t)\, \mathrm{d}t + \alpha^{ij} \beta_u \, \mathrm{d}N_s^j$$

for $i = 1, \dots D$ and $u = 1, \dots U$. This extension of the exponential kernels leads to similar computations but to make the implementation more explicit we detail this more comprehensive formulation. In what follows, we provide explicit calculus for an efficient simulation and the computations of log-likelihood and least squares errors.

Simulation. Thanks to the Markov property, the value of the conditional intensity λ^i at a time t_2 can be obtained from the value at a previous time $t_1 < t_2$ and the events that have occurred between t_1 and t_2. In the particular case for sum of exponentials kernels, we retrieve the intensity λ^i from the relation,

$$\nu^i(t_2|\,\mathscr{F}_{t_2}) = \nu^i(t_1|\,\mathscr{F}_{t_1}) \exp(-\beta_u (t_2 - t_1)) + \sum_{j=1}^{D} \sum_{k : t_1 \le t_k^j < t} \alpha_u^{ij} \beta_u \exp(-\beta_u (t - t_k^j)),$$

where we denote by t_k^j the k-th event of the node j. This relation naturally also holds for exponential kernels. It makes the simulation procedure much more efficient as it is no more required to look each time at all the previous events to evaluate the intensity. The complexity is hence linear instead of quadratic in N_T in the general case.

Log-likelihood. The Markov property also leads to a fast computation of the log-likelihood of Hawkes processes with exponential kernels. The log-likelihood of a multivariate point process is the sum of the log-likelihood expressed in Equation (I.1) over all nodes. For $i = 1, \dots, D$, each term of the sum writes, with the sum of exponentials parametrization,

$$-\log L^i(\mu, \alpha, \mathcal{F}_T) = \mu^i T + \sum_{j=1}^{D} \sum_{u=1}^{U} \alpha_u^{ij} G_u^i - \sum_{k=1}^{N_T^i} \log\Big(\mu^i + \sum_{j=1}^{D} \sum_{u=1}^{U} \alpha_u^{ij} g_u^j(t_k^i)\Big)$$

where

$$g_u^j(t) = \sum_{l : t_l^j < t} \beta_u \exp\big(-\beta_u (t - t_l^j)\big) \quad \text{and} \quad G_u^i = \int_0^T g_u^i(s)\, \mathrm{d}s$$

for $i, j = 1, \dots, D$ and $u = 1, \dots U$. Hence, with all precomputed weights g and G, the computational cost of the likelihood is in $\mathcal{O}(UDN_T)$ where N_T is the total number of events that have occurred across all nodes. This is much better than the complexity of the generic computation which is quadratic in N_T. Then, the Markov property allows the weights g to be computed in linear time using this induction

$$g_u^j(t_k^i) = g_u^j(t_{k-1}^i) \exp(-\beta_u (t_k^i - t_{k-1}^i)) + \sum_{l : t_{k-1}^i \le t_l^j < t_k^i} \beta_u \exp(-\beta_u (t_k^i - t_l^j)),$$

for $k = 1, \dots, N_T^i$, $i, j = 1, \dots, D$ and $u = 1, \dots U$, while the weights G simply writes

$$G_u^i = N_T^i - \sum_{k : t_k^i < T} \exp\big(-\beta_u (T - t_k^i)\big)$$

for $i = 1, \dots D$ and $u = 1, \dots U$. Finally like the likelihood, the cost of the precomputation of the weights is in $\mathcal{O}(UDN_T)$.

Least squares error. The least squares error of a multivariate point process is also the sum of the least square errors (I.2) of the conditional intensities of the D counting processes. It benefits from weights precomputation for a quicker evaluation too. For $i = 1, \dots, D$, each term of the sum writes

$$\begin{aligned} R^i(\mu, \alpha, \mathcal{F}_T) = {\mu^i}^2 T + 2\mu^i \sum_{j=1}^{D} \sum_{u=1}^{U} \alpha_u^{ij} G_u^j + \sum_{j=1}^{D} \sum_{u=1}^{U} \sum_{u'=1}^{U} \alpha_u^{ij} \alpha_{u'}^{ij} K_{uu'}^j \\ + 2 \sum_{j=1}^{D} \sum_{j'=1}^{D} \sum_{u=1}^{U} \sum_{u'=1}^{U} \alpha_u^{ij} \alpha_{u'}^{ij'} \Lambda_{uu'}^{jj'} - 2\mu^i N_T^i - 2 \sum_{j=1}^{D} \sum_{u=1}^{U} \alpha_u^{ij} \sum_{k=1}^{N_T^i} g_u^j(t_k^i), \end{aligned}$$

where the new weights K and Λ are given in Proposition 1. The computation complexity of the weights Λ is $\mathcal{O}(DU^2N_T)$ and is the bottleneck of the precomputation phase. This complexity is U times bigger than for the log likelihood, but once these weights are calculated, the computation of the error has a complexity of $\mathcal{O}(D^3U^2)$ which is independent of the total number of events. Hence, any level of precision can be reached with $D \times U^2$ passes on the data. This makes this error very useful when the number of studied events is very large. Also, once the weights are precomputed this quadratic function is very easy to optimize in contrary to the log-likelihood function where the logarithmic term is problematic (see Chapter IV). Let's detail in the following proposition the writings of the weights K and Λ.

Proposition 1. *For a Hawkes process with sum of exponentials parametrization, the cross term integral can be decomposed in the following two terms*

$$\int_0^T \sum_{j=1}^{D}\sum_{j'=1}^{D}\sum_{u=1}^{U}\sum_{u'=1}^{U} \alpha_u^{ij}\alpha_{u'}^{ij'} g_u^j(t) g_{u'}^{j'}(t)\,\mathrm{d}t$$
$$= \sum_{j=1}^{D}\sum_{u=1}^{U}\sum_{u'=1}^{U} \alpha_u^{ij}\alpha_{u'}^{ij} K_{uu'}^{j} + 2\sum_{j=1}^{D}\sum_{j'=1}^{D}\sum_{u=1}^{U}\sum_{u'=1}^{U} \alpha_u^{ij}\alpha_{u'}^{ij'}\Lambda_{uu'}^{jj'}$$

where $g_u^j(t) = \sum_{l:t_l^j<t} \beta_u \exp\big(-\beta_u(t-t_l^j)\big)$ *for* $j = 1,\dots,D$ *and* $u = 1,\dots,U$ *and the weights* K *and* Λ *are expressed as follow*

$$K_{uu'}^{j} = \sum_{(l:t_l^j<T)} \frac{\beta_u\beta_{u'}}{\beta_u+\beta_{u'}}\Big(1-\exp\big(-(\beta_u+\beta_{u'})(T-t_l^j)\big)\Big)$$

$$\Lambda_{uu'}^{jj'} = \sum_{(l:t_l^j<T)} \frac{\beta_u}{\beta_u+\beta_{u'}} g_{u'}^{j'}(t_l^j)\Big(1-\exp\big(-(\beta_u+\beta_{u'})(T-t_l^j)\big)\Big)$$

for $j, j' = 1,\dots,D$ *and* $u, u' = 1,\dots U$.

Proof. For any function $f:\mathbb{R}^2 \to \mathbb{R}$ symmetric in its two variables, we have the following relation that simply consists in reordering the terms,

$$\sum_{j=1}^{D}\sum_{j'=1}^{D}\sum_{(l:t_l^j<T)}\sum_{(l':t_{l'}^{j'}<T)} f(t_l^j, t_{l'}^{j'}) = \sum_{j=1}^{D}\sum_{j'=1}^{D}\sum_{(l:t_l^j<T)}\sum_{(l':t_{l'}^{j'}\le t_l^j)} f\big(t_l^j, t_{l'}^{j'}\big)\Big(2\cdot\mathbb{1}_{t_{l'}^{j'}<t_l^j} + \mathbb{1}_{t_{l'}^{j'}=t_l^j}\Big).$$

As all timestamps are distinct, the indicator function $\mathbb{1}_{t_{l'}^{j'}=t_l^j}$ more simply rewrites $\mathbb{1}_{j=j',l=l'}$. Here, the weights K correspond to the terms $\mathbb{1}_{t_{l'}^{j'}=t_l^j}$ of the sum, Λ to the terms $\mathbb{1}_{t_{l'}^{j'}<t_l^j}$, and f is set to

$$f(t_l^j, t_{l'}^{j'}) = \sum_{u=1}^{U}\sum_{u'=1}^{U} \beta_u \exp\big(-\beta_u(t-t_l^j)\big)\beta_{u'}\exp\big(-\beta_{u'}(t-t_{l'}^{j'})\big).$$

If we introduce,

$$H_{u,u'}\big(t_l^j, t_{l'}^{j'}\big) = \int_{t_l^j}^{T} \beta_u \exp\big(-\beta_u (t - t_l^j)\big) \beta_{u'} \exp\big(-\beta_{u'}(t - t_{l'}^{j'})\big)\, \mathrm{d}t$$
$$= \frac{\beta_u \beta_{u'}}{\beta_u + \beta_{u'}} \exp\big(-\beta_{u'}(t_l^j - t_{l'}^{j'})\big)\Big(1 - \exp\big(-(\beta_u + \beta_{u'})(T - t_l^j)\big)\Big)$$

for $u, u' = 1, \dots D$, both weights K and Λ can be expressed as

$$K_{uu'}^j = \sum_{(l : t_l^j < T)} H_{u,u'}\big(t_l^j, t_l^j\big) \quad \text{and} \quad \Lambda_{uu'}^{jj'} = \sum_{(l : t_l^j < T)} \sum_{(l' : t_{l'}^{j'} < t_l^j)} H_{u,u'}\big(t_l^j, t_{l'}^{j'}\big)$$

for $i, j, j' = 1, \dots, D$ and $u, u' = 1, \dots U$. Finally, the writing of the weights Λ can be optimized using the precomputed weights g leading to the expression given in the proposition statement. ■

In the following example, we use the `tick` library (see Chapter V) to highlight the properties of the least-squares goodness-of-fit. First, we simulate a 30 nodes Hawkes process whose kernels are parametrized with sum of exponentials. Then we retrieve the generating parameters by minimizing the least squares error. Finally, we evaluate the estimated parameters by comparing them to the generating one with the estimation error, namely

$$\text{Estimation error} = \sum_{i,j=1}^{D} \sum_{u=1}^{U} \big(\alpha_u^{ij} - \hat{\alpha}_u^{ij}\big)^2,$$

where $\hat{\alpha}_u^{ij}$ is the estimated value of α_u^{ij} for $i, j = 1, \dots, D$ and $u = 1, \dots U$.

```python
from tick.hawkes import SimuHawkesSumExpKernels, HawkesSumExpKern

# Parameters
n_nodes, n_decays = 30, 3
baselines = np.random.rand(n_nodes) / n_nodes
decays = np.random.rand(n_decays)
adjacency = np.random.rand(n_nodes, n_nodes, n_decays)

# Simulation
hawkes = SimuHawkesSumExpKernels(
    baseline=baselines, decays=decays, adjacency=adjacency)
hawkes.adjust_spectral_radius(0.8)
hawkes.max_jumps = 100000
hawkes.simulate()

# Estimation
learner = HawkesSumExpKern(decays=decays, tol=1e-4)
learner.fit(hawkes.timestamps)

estimation_error = np.linalg.norm(hawkes.adjacency - learner.adjacency)**2
```

In Figure I.2, this experiment has been repeated multiple times with different settings. First, different sub-samples of 30 000, 100 000, 300 000 and 1 000 000 events have been considered leading to more or less big datasets. Second, we vary the precision levels, meaning that we stop the optimization procedure when the gain in term of least square error becomes lower than this level. To present the results, we have split the fitting time into the weights computation and the optimization timings. Figure I.2 illustrates several expected behaviors. To begin with, while the weights computation time (a) grows with the number of events, the optimization time (b) depends only on the precision level and is not the bottleneck anymore when the sample is big (see sample 1000K in which the weights computation step is much longer than the optimization). Moreover, for all precision levels, the estimation error (c) gets lower when the sample size increases. However, low precisions procedures cannot reach the best estimation errors. For example, in (c) with a 10^{-2} (resp. 10^{-4}) precision level we cannot reach an estimation error lower than 0.4 (resp. 0.2). Finally, even if reaching high precision levels is not very costly, it can though lead to overfitting as illustrated in (c) where level 10^{-6} has poor performance on the smallest sample 30K but has the best ones on the biggest sample 1000K.

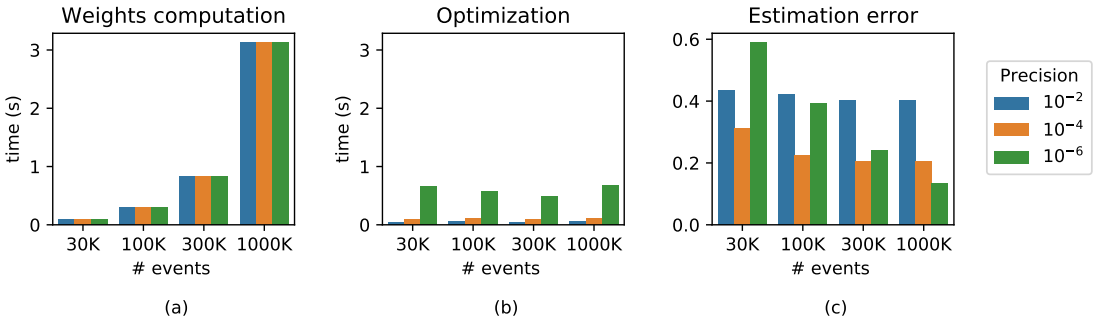

Figure I.2: Timings and estimation errors for various sample sizes and precision levels. This figure illustrates that the optimization phase in $\mathcal{O}(D^3U^2)$ is independent on the sample size N_T (b) and can be cheaper than the precomputation phase (a). It also highlights the overfitting possibility while combining high precision levels with too small datasets (c).

CHAPTER II

Sparse and low-rank multivariate Hawkes processes

Abstract

We consider the problem of unveiling the implicit network structure of node interactions (such as user interactions in a social network), based only on high-frequency timestamps. Our inference is based on the minimization of the least-squares loss associated with a multivariate Hawkes model, penalized by ℓ_1 and trace norm of the interaction tensor. We provide a first theoretical analysis for this problem, that includes sparsity and low-rank inducing penalizations. This result involves a new data-driven concentration inequality for matrix martingales in continuous time with observable variance, which is a result of independent interest and a broad range of possible applications since it extends to matrix martingales former results restricted to the scalar case. A consequence of our analysis is the construction of sharply tuned ℓ_1 and trace-norm penalizations, that leads to a data-driven scaling of the variability of information available for each users. Numerical experiments illustrate the significant improvements achieved by the use of such data-driven penalizations.

1 Introduction

Understanding the dynamics of social interactions is a challenging problem of rapidly growing interest [dMB04, Les08, CS08, LBK09] because of the large number of applications in web-advertisement and e-commerce, where large-scale logs of event history are available. A common supervised approach consists in the prediction of labels based on declared interactions (friendship, like, follower, etc.). However such supervision is not always available, and it does not always describe accurately the level of interactions between users. Labels are often only binary while a quantification of

the interaction is more interesting, declared interactions are often deprecated, and more generally a supervised approach is not enough to infer the latent communities of users, as temporal patterns of actions of users are much more informative.

For latent social groups recovering, several recent papers [RBS11, GRLS13, DRSS14] consider an approach directly based on the real *actions* or *events* of users (referred to as *nodes* in the following) that are fully identified through their corresponding user id and timestamp. These models assume a structure of data consisting in a sequence of independent cascades, containing the timestamp of each node. In these works, techniques coming from survival analysis are used to derive a tractable convex likelihood, that allows one to infer the latent community structure. However, they require that data are already segmented into sets of independent cascades, which is often unrealistic. Moreover, it does not allow for recurrent events, namely a node can be infected only once, and it cannot incorporate exogenous factors, i.e., influence from the world outside the network.

Another approach is based on self-exciting point processes, such as the Hawkes process [Haw71b]. Previously used for geophysics [Oga98], high-frequency finance [BDHM13, BMM15], crime activity [MSB+11], these processes have been recently used for the modelization of users activity in social networks, see for instance [CS08, BBH12, ZZS13a, YZ13]. The structure of the Hawkes model allows us to capture the direct influence of a specific user's action on all the future actions of all the users (including himself). It encompasses in a single likelihood the decay of the influence over time, the levels of interaction between nodes, which can be seen as a weighted asymmetrical adjacency matrix, and a baseline intensity, that measures the level of exogeneity of a user, namely the spontaneous apparition of an action, with no influence from other nodes of the network.

In this paper, we consider such a multivariate Hawkes process (MHP), and we combine convex proxies for sparsity and low-rank of the adjacency tensor and the baseline intensities, that are now of common use in low-rank modeling in collaborative filtering problems [CT04, CT10]. Note that this approach is also considered in [ZZS13a]. We provide a first theoretical analysis of the generalization error for this problem, see [HRBR15] for an analysis including only entrywise ℓ_1 penalization. Namely, we prove a sharp oracle inequality for our procedure, that includes sparsity and low-rank inducing priors, see Theorem 4 in Section 5. This result involves a new data-driven concentration inequality for matrix martingales in continuous time, see Theorems 2 and 3 in Section 3.3, that are results of independent interest, that extends previous non-commutative versions of concentration inequalities for martingales in discrete time, see [Tro12]. A consequence of our analysis is the construction of sharply tuned ℓ_1 and trace-norm penalizations, that leads to a data-driven scaling of the variability of information available for each node. We give empirical evidence of the improvements of our data-driven penalizations, by conducting in Section 6 nu-

merical experiments on simulated data. Since the objectives involved are convex with a smooth component, our algorithms build upon standard batch proximal gradient descent algorithms.

2 The multivariate Hawkes model and the least-squares functional

Consider a finite network with d nodes (each node corresponding to a user in a social network for instance). For each node $j \in \{1,\dots,d\}$, we observe the timestamps $\{t_{j,1}, t_{j,2},\dots\}$ of actions of node j on the network (a message, a click, etc.). With each node j is associated a counting process $N_j(t) = \sum_{i\geq 1} \mathbb{1}_{t_{j,i}\leq t}$ and we consider the d-dimensional counting process $N_t = [N_1(t) \cdots N_d(t)]^\top$, for $t \geq 0$. We observe this process for $t \in [0,T]$. Each N_j has an intensity λ_j, meaning that

$$\mathbb{P}\big(N_j \text{ has a jump in } [t, t+dt] \mid \mathscr{F}_t\big) = \lambda_j(t)dt, \quad j = 1,\dots,d,$$

where $\mathscr{F}_t$ is the σ-field generated by N up to time t. The multivariate Hawkes model assumes that each N_j has an intensity $\lambda_{j,\theta}$ given by

$$\lambda_{j,\theta}(t) = \mu_j + \sum_{j'=1}^{d} \int_{(0,t)} \varphi_{j,j'}(t-s)\,\mathrm{d}N_{j'}(s), \tag{II.1}$$

where $\mu_j \geq 0$ is the baseline intensity of j (i.e., the intensity of exogenous events of node j) and where the functions $\varphi_{j,j'} : \mathbb{R}^+ \to \mathbb{R}$ for $j = 1,\dots,d$, called *kernels*, allow to quantify the impact of node j' on node j. Note that the integral used in Equation (II.1) is a Stieljes integral, namely it simply stands for

$$\int_{(0,t)} \varphi(t-s)\,\mathrm{d}N_{j'}(s) = \sum_{i\,:\,t_{j',i}\in[0,t)} \varphi(t-t_{j',i}).$$

In the paper, we consider general kernel functions $\varphi_{j,j'}(t)$ that can be written as:

$$\varphi_{j,j'}(t) = \sum_{k=1}^{K} a_{j,j',k} h_{j,j',k}(t). \tag{II.2}$$

where the coefficients $a_{j,j',k}$ are the entries of a $d \times d \times K$ tensor $\mathbb{A}$ (i.e., $(\mathbb{A})_{j,j',k} = a_{j,j',k}$) and the kernels $h_{j,j',k}(t)$ are elements of a fixed dictionnary of non negative and causal functions ($h_{j,j',k} : \mathbb{R}^+ \to \mathbb{R}^+$) such that $\|h_{j,j',k}\|_1 = 1$. In that respect, the weights $a_{j,j',1},\dots,a_{j,j',K}$ all quantify the influence of j' on j, but the particular weight $a_{j,j',k}$ quantifies it for the k-th *decay function* $h_{j,j',k}$. A standard choice is a

dictionnary of exponential kernels, $h_{j,j',k}(t) = \alpha_k e^{-\alpha_k t}$ with varying memory parameters $\alpha_1, \dots, \alpha_K$. This leads to the following standard parametrization of the kernel functions, called *exponential kernels*:

$$\varphi_{j,j'}(t) = \sum_{k=1}^{K} a_{j,j',k} \alpha_k \exp(-\alpha_k t). \tag{II.3}$$

The main advantage of exponential kernels with fixed memory parameters $\alpha_1, \dots, \alpha_K$, is that it allows one to handle a convex problem. In the general case or when the memory parameters are unknown, the problem becomes non-convex, more challenging and is beyond the scope of the paper.

The parameter of interest is the *self-excitement* tensor $\mathbb{A}$, which can be viewed as a cross-scale (for $k = 1, \dots, K$) weighted adjacency matrix of connectivity between nodes, as illustrated in Figure II.1 below.

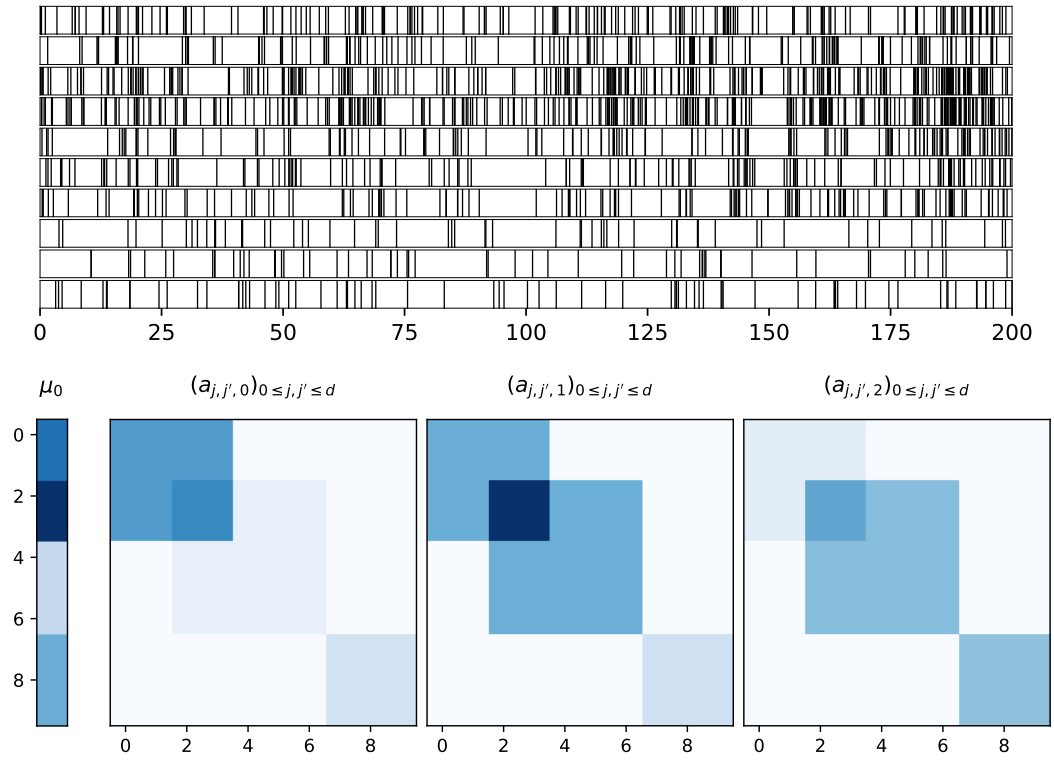

Figure II.1: Toy example with $d = 10$ nodes. Based on actions' timestamps of the nodes, represented by vertical bars (top), we aim at recovering the vector μ_0 and the tensor $\mathbb{A}$ of implicit influence between nodes (bottom).

The Hawkes model is particularly relevant for the modelization of the "microscopic" activity of social networks and has attracted a lot of interest in the recent

literature (see [CS08, BBH12, ZZS13a, YZ13, LA14, DBS13, BBH12, ISG13], among others) for this kind of application, with a particular emphasis on [HRBR15] that gives first theoretical results for the Lasso used with Hawkes processes with an application to neurobiology. The main point is that this simple autoregressive structure of the intensity allows us to capture the direct influence of a user, based on the recurrence and the patterns of his actions, by separating the intensity into a baseline and a self-exciting component, hence allowing to filter exogeneity in the estimation of users' influences on each others.

We introduce in this paper an estimation procedure of $\theta = (\mu, \mathbb{A})$ based on data $\{N_t : t \in [0, T]\}$. The hidden structure underlying the observed actions of nodes is contained in $\mathbb{A}$. Our strategy is based on the least-squares functional given by

$$R_T(\theta) = \|\lambda_\theta\|_T^2 - \frac{2}{T}\sum_{j=1}^{d}\int_{[0,T]} \lambda_{j,\theta}(t)\,\mathrm{d}N_j(t), \tag{II.4}$$

with respect to θ, where $\|\lambda_\theta\|_T^2 = \frac{1}{T}\sum_{j=1}^{d}\int_{[0,T]}\lambda_{j,\theta}(t)^2\,\mathrm{d}t$ is the norm associated with the inner product

$$\langle \lambda_\theta, \lambda_{\theta'}\rangle_T = \frac{1}{T}\sum_{j=1}^{d}\int_{[0,T]}\lambda_{j,\theta}(t)\lambda_{j,\theta'}(t)\,\mathrm{d}t. \tag{II.5}$$

This least-squares function is very natural, and comes from the empirical risk minimization principle [VDG00, Mas07, Koll1, BM06]: assuming that N_j has an unknown ground truth intensity λ_j (not necessarily following the Hawkes model), the Doob-Meyer's decomposition gives

$$\int_{[0,T]}\lambda_{j,\theta}(t)\,\mathrm{d}N_j(t) = \int_{[0,T]}\lambda_{j,\theta}(t)\lambda(t)\,\mathrm{d}t + \int_{[0,T]}\lambda_{j,\theta}(t)\,\mathrm{d}M_j(t),$$

where $M_j(t) = N_j(t) - \int_0^t \lambda(s)\,\mathrm{d}s$ is a continuous-time martingale with upwards jumps of +1. Since the "noise" term $\int_{[0,T]}\lambda_{j,\theta}(t)\,\mathrm{d}M_j(t)$ is centered, we obtain

$$\mathbb{E}[R_T(\theta)] = \mathbb{E}\|\lambda_\theta\|_T^2 - 2\mathbb{E}\langle\lambda_\theta, \lambda\rangle_T = \mathbb{E}\|\lambda_\theta - \lambda\|_T^2 - \|\lambda\|_T^2,$$

so that we expect a minimizer $\hat{\theta}$ of $R_T(\theta)$ to lead to a good estimation $\lambda_{\hat{\theta}}$ of λ, following the empirical risk minimization principle. As explained in Section 8 below, the noise terms can be written as

$$\int_0^t \mathbb{T}_s \circ \mathrm{d}\boldsymbol{M}_s,$$

for a specific tensor $\mathbb{T}_t$ and matrix martingale $\boldsymbol{M}_t$, where $\mathbb{T}_s \circ \boldsymbol{M}_s$ stands for a tensor-matrix product defined in Section 3.1 below. The next Section introduces new results, of independent interest, providing *data-driven* deviation inequalities for the operator

norm of a matrix martingale defined as the stochastic integral $\int_0^t \mathbb{T}_s \circ \mathrm{d}\boldsymbol{M}_s$. These results allow us, as a by-product, to control the noise terms arising in the application considered in this paper, and lead to a sharp data-driven tuning of the penalizations used on $\mathbb{A}$, as explained in Section 4 below.

3 A new data-driven matrix martingale Bernstein's inequality

An important ingredient for the theoretical results proposed in this paper is an observable deviation inequality for continuous time matrix martingales. We first recall previous results obtained in [BGM18] about non-observable deviation inequalities for such objects.

3.1 Notations

Let $\mathbb{T}$ be a tensor of shape $m \times n \times p \times q$. It can be considered as a linear mapping from $\mathbb{R}^{p\times q}$ to $\mathbb{R}^{m\times n}$ according to the following "tensor-matrix" product:

$$(\mathbb{T} \circ \boldsymbol{A})_{i,j} = \sum_{k=1}^{p}\sum_{l=1}^{q} \mathbb{T}_{i,j;k,l}\boldsymbol{A}_{k,l}.$$

We will denote by $\mathbb{T}^\top$ the tensor such that $\mathbb{T}^\top \circ \boldsymbol{A} = (\mathbb{T} \circ \boldsymbol{A})^\top$ (i.e., $\mathbb{T}^\top_{i,j;k,l} = \mathbb{T}_{j,i;k,l}$) and by $\mathbb{T}_{\bullet,\bullet;k,l}$ and $\mathbb{T}_{i,j;\bullet,\bullet}$ the matrices obtained when fixing the indices k,l and i,j respectively. Note that $(\mathbb{T} \circ \boldsymbol{A})_{i,j} = \mathrm{tr}(\mathbb{T}_{i,j;\bullet,\bullet}\boldsymbol{A}^\top)$. If $\mathbb{T}$ and $\mathbb{T}'$ are two tensors of dimensions $m\times n\times p\times q$ and $n\times r\times p\times q$ respectively, $\mathbb{T}\mathbb{T}'$ stands for the $m\times r\times p\times q$ tensor defined as $(\mathbb{T}\mathbb{T}')_{i,j;k,l} = (\mathbb{T}_{\bullet,\bullet;k,l}\mathbb{T}'_{\bullet,\bullet;k,l})_{i,j}$. Accordingly, for an integer $r \geq 1$, if $\mathbb{T}_{\bullet,\bullet;a,b}$ are square matrices, we will denote by $\mathbb{T}^r$ the tensor such that $(\mathbb{T}^r)_{i,j;k,l} = (\mathbb{T}^r_{\bullet,\bullet;k,l})_{i,j}$. We also introduce $\|\mathbb{T}\|_{\mathrm{op};\infty} = \max_{k,l}\|\mathbb{T}_{\bullet,\bullet;k,l}\|_{\mathrm{op}}$, the maximum operator norm of all matrices formed by the first two dimensions of tensor $\mathbb{T}$.

In this paper we shall consider the class of $m\times n$ matrix martingales that can be written as

$$\boldsymbol{Z}_{\mathbb{T}}(t) = \int_0^t \mathbb{T}_s \circ \mathrm{d}\boldsymbol{M}_s, \tag{II.6}$$

where $\mathbb{T}_s$ is a tensor with dimensions $m\times n\times p\times q$, whose components are assumed to be locally bounded predictable random functions. The process $\boldsymbol{M}_t$ is a $p\times q$ matrix with entries that are square integrable martingales with a diagonal quadratic covariation matrix. More explicitly, the entries of $\boldsymbol{Z}_{\mathbb{T}}(t)$ are given by

$$(\boldsymbol{Z}_{\mathbb{T}}(t))_{i,j} = \sum_{k=1}^{p}\sum_{l=1}^{q}\int_0^t (\mathbb{T}_s)_{i,j;k,l}(\mathrm{d}\boldsymbol{M}_s)_{k,l},$$

where the martingale $\boldsymbol{M}_t$ is a matrix of compensated counting processes $\boldsymbol{M}_t = \boldsymbol{N}_t - \boldsymbol{\lambda}_t$ where $\boldsymbol{N}_t$ is a $p \times q$ matrix counting process (i.e., each component is a counting process) with an intensity process $\boldsymbol{\lambda}_t$ which is predictable, continuous and with finite variations (FV).

3.2 A non-observable matrix martingale Bernstein's inequality

The next Theorem (which is a small variation of Theorem 2 in [BGM18]) provides a concentration inequality for $\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}}$, the operator norm of $\boldsymbol{Z}_{\mathbb{T}}(t)$. Before stating the Theorem, let us introduce some more notations. We define

$$b_{\mathbb{T}}(t) = \sup_{0 \le s \le t} \max\big(\|\mathbb{T}_s\|_{\mathrm{op};\infty}, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty}\big),$$

and depending on whether the tensor $\mathbb{T}_s$ is symmetric (i.e., $\mathbb{T}_s^\top = \mathbb{T}_s$ and $m = n$) or not, we define the following.

- If $\mathbb{T}_s$ is symmetric, we define

$$\boldsymbol{W}_{\mathbb{T}}(s) = \mathbb{T}_s^2 \circ \boldsymbol{\lambda}_s$$

  and $K_{m,n} = m$

- If $\mathbb{T}_s$ is not symmetric, we define

$$\boldsymbol{W}_{\mathbb{T}}(s) = \begin{bmatrix} \mathbb{T}_s \mathbb{T}_s^\top \circ \boldsymbol{\lambda}_s & \boldsymbol{0} \\ \boldsymbol{0} & \mathbb{T}_s^\top \mathbb{T}_s \circ \boldsymbol{\lambda}_s \end{bmatrix}, \tag{II.7}$$

  and $K_{m,n} = m + n$.

In both cases, we define

$$\boldsymbol{V}_{\mathbb{T}}(t) = \int_0^t \boldsymbol{W}_{\mathbb{T}}(s)\, \mathrm{d}s.$$

Finally, all along the paper we denote $\phi(x) = e^x - 1 - x$ for $x \in \mathbb{R}$. The following concentration inequality is an easy consequence of Theorem 1 from [BGM18] where $\lambda_{\max}(A)$ denotes the largest eigenvalue of the matrix A.

Theorem 1. *Let $\boldsymbol{Z}_{\mathbb{T}}(t)$ be the $m \times n$ matrix martingale given by Equation* (II.6). *Moreover, assume that*

$$\mathbb{E}\Big[\int_0^t \frac{\phi\big(3\max(\|\mathbb{T}_s\|_{\mathrm{op};\infty}, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty})\big)}{\max(\|\mathbb{T}_s\|_{\mathrm{op};\infty}^2, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty}^2)} (\boldsymbol{W}_{\mathbb{T}}(s))_{i,j}\, \mathrm{d}s\Big] < +\infty,$$

for any $1 \le i, j \le m+n$. *Then for any* $\xi \in (0,3)$, $t, b, x > 0$, *the following holds*:

$$\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge \frac{\phi(\xi)}{\xi b} \lambda_{\max}\big(\boldsymbol{V}_{\mathbb{T}}(t)\big) + \frac{xb}{\xi}, \quad b_{\mathbb{T}}(t) \le b \Big] \le K_{m,n} e^{-x}. \tag{II.8}$$

Optimizing this last inequality on ξ *gives*

$$\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge \sqrt{2vx} + \frac{bx}{3}, \lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) \le v, \quad b_{\mathbb{T}}(t) \le b \Big] \le K_{m,n} e^{-x}. \tag{II.9}$$

The proof of Theorem 1 is given in Section 8.1 below. This result is a Freedman (or Bernstein) inequality for the operator norm of $\boldsymbol{Z}_{\mathbb{T}}(t)$, that provides a deviation based on a variance term $\boldsymbol{V}_{\mathbb{T}}(t)$ and a L^∞ term $b_{\mathbb{T}}(t)$. It is a strong generalization of the scalar Freedman inequality for continuous time martingales, and this result match exactly the scalar case whenever $\boldsymbol{Z}_{\mathbb{T}}(t)$ is scalar. A more thorough discussion about the consequences of this result is provided in [BGM18].

3.3 Data-driven matrix martingale Bernstein's inequalities

Inequality (II.9) is of poor practical interest in situations where one observes only the jumping times of the $\boldsymbol{Z}_t$ components (namely $\boldsymbol{N}_t$) and not the stochastic intensity $\boldsymbol{\lambda}_t$. In that respect, one needs a "data driven" inequality where $\boldsymbol{V}_{\mathbb{T}}(t)$ is replaced by its empirical version $\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)$.

- If $\mathbb{T}_s$ is symmetric, we define

$$\widehat{\boldsymbol{V}}_{\mathbb{T}}(t) = \int_0^t \mathbb{T}_s^2 \circ \mathrm{d}\boldsymbol{N}_s,$$

- while if $\mathbb{T}_s$ is not symmetric, we define

$$\widehat{\boldsymbol{V}}_{\mathbb{T}}(t) = \begin{bmatrix} \int_0^t \mathbb{T}_s \mathbb{T}_s^\top \circ \mathrm{d}\boldsymbol{N}_s & \boldsymbol{0} \\ \boldsymbol{0} & \int_0^t \mathbb{T}_s^\top \mathbb{T}_s \circ \mathrm{d}\boldsymbol{N}_s \end{bmatrix}.$$

The next Proposition allows us to control $\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t))$ using its observable counterpart $\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))$ with a large probability. This result is a generalization to arbitrary matrices of dimensions $m \times n$ of an analog inequality originally proven by Hansen et al. [HRBR15] for scalar martingales.

Proposition 1. *For any* $x, b > 0$ *and* $\xi \in (0,3)$ *such that* $\xi > \phi(\xi)$, *we have*

$$\mathbb{P}\Big[\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) \ge \frac{\xi}{\xi - \phi(\xi)} \lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) + \frac{xb^2}{\xi - \phi(\xi)}, \quad b_{\mathbb{T}}(t) \le b \Big] \le K_{m,n} e^{-x},$$

where $K_{m,n}$ is defined as in Theorem 1. Moreover, choosing $\xi = -W_{-1}(-\frac{2}{3}e^{-2/3}) - 2/3$ (note that $\xi \approx 0.762$), where W_{-1} is the second branch of the Lambert W function, leads to

$$\mathbb{P}\Big[\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) \geq 2\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) + cb^2x, \quad b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x}$$

for any $x, b > 0$, with $c = 2.62$.

Thanks to Proposition 1, we can establish an analog of Theorem 1 where $\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t))$ is replaced by its data-driven version $\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))$, up to a slight loss in values of the numerical constants.

Theorem 2. *With the same notations and assumptions as in Theorem 1 one has*

$$\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \geq 2\sqrt{vx} + cbx,\ \lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \leq v,\ b_{\mathbb{T}}(t) \leq b\Big] \leq 2K_{m,n}e^{-x} \tag{II.10}$$

for any $x, b > 0$ with $c = 14.39$.

The proof of Theorem 2 is given in Section 8.3 below. It follows simple arguments that combine Theorem 1 and Proposition 1. However, this inequality is stated on the events $\{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \leq v\}$ and $\{b_{\mathbb{T}}(t) \leq b\}$, while an unconditional deviation inequality is more practical. Such a result, which involves some extra technicalities, is stated in the next Theorem.

Theorem 3. *With the same conditions and notations as in Theorem 2, one has*

$$\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \geq 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(x + \ell_x(t))} + c(x + \ell_x(t))(1 + b_{\mathbb{T}}(t))\Big] \leq C_{m,n}e^{-x} \tag{II.11}$$

where $C_{m,n} = \frac{\pi^4}{18\log(2)^4}K_{m,n} \leq 23.45K_{m,n}$, where $c = 14.39$ and

$$\ell_x(t) = 2\log\log\Big(\frac{4\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))}{x} \vee 2\Big) + 2\log\log(4b_{\mathbb{T}}(t) \vee 2).$$

The proof of this Theorem is given in Section 8.4. It is a result of independent interest, that gives a control on the operator norm of a matrix martingale in continuous time (with jumps at most 1), using only observable quantities. Along with [BGM18], it provides a first deviation inequality for such objects, and it can be understood as a data-driven version of the results given in [BGM18].

4 The procedure

We want to produce an estimation procedure of $\theta = (\mu, \mathbb{A})$ based on data from $\{N_t : t \in [0, T]\}$. Following the empirical risk minimization principle, the estimation

procedure uses the least-squares functional (II.4) as a goodness-of-fit. In addition to this goodness-of-fit criterion, we need to use a penalization that allows us to reduce the dimensionality of the model, namely we consider

$$\hat{\theta} \in \operatorname*{arg\,min}_{\theta=(\mu,\mathbb{A})\in\mathbb{R}_+^d\times\mathbb{R}_+^{d\times d\times K}} \{R_T(\theta) + \mathrm{pen}(\theta)\}, \tag{II.12}$$

for a specific penalization function $\mathrm{pen}(\theta)$ described below. In particular, we want to reduce the dimensionality of $\mathbb{A}$, based on the prior assumption that latent factors explain the connectivity of users in the network. This leads to a low-rank assumption on $\mathbb{A}$, which is commonly used in collaborative filtering and matrix completion techniques [RRS11]. Our prior assumptions on μ and $\mathbb{A}$ are the following.

Sparsity of μ. Some nodes are basically inactive and react only if stimulated. Hence, we assume that the baseline intensity vector μ is sparse.

Sparsity of $\mathbb{A}$. A node interacts only with a fraction of other nodes, meaning that for a fixed node j, only a few $a_{j,j',k}$ are non-zero. Moreover, a node might react at specific time scales only, namely $a_{j,j',k}$ is non-zero for some k only for fixed j, j'. Hence, we assume that $\mathbb{A}$ is an entrywise sparse tensor.

Low-rank of $\mathbb{A}$. Using together Equations (II.1) and (II.2), one can write

$$\begin{aligned}\lambda_{j,\theta}(t) &= \mu_j + \sum_{j'=1}^{d}\sum_{k=1}^{K} a_{j,j',k}\int_{(0,t)} h_{j,j',k}(t-s)\,\mathrm{d}N_{j'}(s)\\ &= \mu_j + \big(\mathrm{hstack}(\mathbb{A})_{j,\bullet}\big)^\top \mathrm{hstack}(\mathbb{H}(t))_{j,\bullet},\end{aligned} \tag{II.13}$$

where $\mathbb{H}(t)$ is the $d\times d\times K$ tensor with entries

$$\mathbb{H}_{j,j',k}(t) = \int_{(0,t)} h_{j,j',k}(t-s)\,\mathrm{d}N_{j'}(s), \tag{II.14}$$

where $(\boldsymbol{X})_{j,\bullet}$ stands for the j-th row of a matrix $\boldsymbol{X}$ and where hstack stands for the horizontally stacking operator defined by

$$\mathrm{hstack}:\mathbb{R}^{d\times d\times K}\to\mathbb{R}^{d\times Kd} \quad \text{such that} \quad \mathrm{hstack}(\mathbb{A}) = \begin{bmatrix}\mathbb{A}_{\bullet,\bullet,1} & \cdots & \mathbb{A}_{\bullet,\bullet,K}\end{bmatrix}, \tag{II.15}$$

where $\mathbb{A}_{\bullet,\bullet,k}$ stands for the $d\times d$ matrix with entries $(\mathbb{A}_{\bullet,\bullet,k})_{j,j'} = \mathbb{A}_{j,j',k}$. In view of Equation (II.13), all the impacts of nodes j' at time scale k on node j is encoded in the j-th row of the $d\times Kd$ matrix $\mathrm{hstack}(\mathbb{A})$. Therefore, a natural assumption is that the matrix $\mathrm{hstack}(\mathbb{A})$ has a low-rank: we assume that there exist latent factors that explain the way nodes impact other nodes through the different scales $k = 1,\dots,K$.

To induce these prior assumptions on the parameters, we use a penalization based on a mixture of the ℓ_1 and trace-norms. These norms are respectively the tightest convex relaxations for sparsity and low-rank, see for instance [CT04, CT10]. They provide state-of-the art results in compressed sensing and collaborative filtering problems, among many other problems. These two norms have been previously combined for the estimation of sparse and low-rank matrices, see for instance [RGV14] and [ZZS13a] in the context of MHP. Therefore, we consider the following penalization on the parameter $\theta = (\mu, \mathbb{A})$:

$$\mathrm{pen}(\theta) = \|\mu\|_{1,\hat{w}} + \|\mathbb{A}\|_{1,\hat{\mathbb{W}}} + \hat{\tau}\,\|\mathrm{hstack}(\mathbb{A})\|_*, \tag{II.16}$$

where each terms are entrywise weighted ℓ_1 and trace-norm penalizations given by

$$\|\mu\|_{1,\hat{w}} = \sum_{j=1}^{d} \hat{w}_j |\mu_j|, \quad \|\mathbb{A}\|_{1,\hat{\mathbb{W}}} = \sum_{1\le j,j'\le d, 1\le k\le K} \hat{\mathbb{W}}_{j,j',k} |\mathbb{A}_{j,j',k}|, \quad \|\boldsymbol{A}\|_* = \sum_{j=1}^{d} \sigma_j(\boldsymbol{A}),$$

where the $\sigma_1(\boldsymbol{A}) \ge \cdots \ge \sigma_d(\boldsymbol{A})$ are the singular values of a matrix $\boldsymbol{A}$ (we take $\boldsymbol{A} = \mathrm{hstack}(\mathbb{A})$ in the penalization). The weights $\hat{w}$, $\hat{\mathbb{W}}$, and coefficients $\hat{\tau}$ are data-driven tuning parameters described below. The choice of these weights comes from a sharp analysis of the noise terms and lead to a data-driven scaling of the variability of information available for each nodes.

From now on, we fix some confidence level $x > 0$, which corresponds to the probability that the oracle inequality from Theorem 4 holds (see Section 5 below). This can be safely chosen as $x = \log T$ for instance, as described in our numerical experiments (see Section 6 below).

Weight $\hat{\tau}$ for the trace-norm penalization of $\mathrm{hstack}(\mathbb{A})$. This weight comes from Corollary 1 (see Section 8.5). Let us introduce the $d \times Kd$ matrix $\boldsymbol{H}(t) = \mathrm{hstack}(\mathbb{H}(t))$ where $\mathbb{H}(t)$ is the $d \times d \times K$ tensor defined by (II.14) and hstack is the horizontally stacking operator defined by (II.15). Let us also recall that $\|\cdot\|_2$ is the ℓ_2-norm, and define $\|\boldsymbol{H}\|_{\infty,2} = \max_{1\le j\le d} \|\boldsymbol{H}_{j,\bullet}\|_2$ where $\boldsymbol{H}_{j,\bullet}$ stands for the j-th row of $\boldsymbol{H}$. We define

$$\begin{aligned}\hat{\tau} = 4\sqrt{\frac{\lambda_{\max}(\widehat{\boldsymbol{V}}(T)/T)(x + \log(2d) + \ell_\tau(T))}{T}} \\ + 28.78\frac{x + \log(2d) + \ell_\tau(T))(1 + \sup_{0\le t\le T}\|\boldsymbol{H}(t)\|_{\infty,2})}{T}\end{aligned} \tag{II.17}$$

where

$$\lambda_{\max}(\widehat{\boldsymbol{V}}(T)) = \lambda_{\max}\Big(\int_0^T \boldsymbol{H}^\top(s)\boldsymbol{H}(s)\,\mathrm{diag}(\mathrm{d}N(s))\Big) \bigvee \max_{j=1,\dots,d} \int_0^T \|\boldsymbol{H}_{j,\bullet}(t)\|_2^2\,\mathrm{d}N_j(s),$$

and where

$$\ell_\tau(T) = 2\log\log\Big(\frac{4\lambda_{\max}(\widehat{\boldsymbol{V}}(T))}{x} \vee 2\Big) + 2\log\log\Big(4 \sup_{0\le t\le T} \|\boldsymbol{H}(t)\|_{\infty,2} \vee 2\Big),$$

where we used the notation $a \vee b = \max(a, b)$ for $a, b \in \mathbb{R}$.

Weights $\hat{w}_j$ for ℓ_1-penalization of μ. These weights are given by

$$\hat{w}_j = 6\sqrt{\frac{(N_j(T)/T)(x + \log d + \ell_j(T))}{T}} + 86.34\frac{x + \log d + \ell_j(T)}{T} \tag{II.18}$$

with $\ell_j(T) = 2\log\log(\frac{4N_j(T)}{x} \vee 2)) + 2\log\log 4$. The weighting of each coordinate j in the penalization of μ is natural: it is roughly proportional to the square-root of $N_j(T)/T$, which is the average intensity of events on coordinate j. The term $\ell_j(T)$ is a technical term, that can be neglected in practice, see Section 6.

Weights $\hat{\mathbb{W}}_{j,j'k}$ for ℓ_1-penalization of $\mathbb{A}$. Recall that the tensor $\mathbb{H}$ is given by (II.14). The weights $\hat{\mathbb{W}}_{j,j'k}$ are given by

$$\begin{aligned}\hat{\mathbb{W}}_{j,j',k} = 4\sqrt{\frac{\frac{1}{T}\int_0^T \mathbb{H}_{j,j',k}(t)^2 \,\mathrm{d}N_j(t)\,(x + \log(Kd^2) + \mathbb{L}_{j,j',k}(T))}{T}} \\ + 28.78\frac{(x + \log(Kd^2) + \mathbb{L}_{j,j',k}(T))(1 + \sup_{0\le t\le T}|\mathbb{H}_{j,j',k}(t)|)}{T}\end{aligned} \tag{II.19}$$

where $\mathbb{L}_{j,j',k}(T) = 2\log\log\big(\frac{4\int_0^T \mathbb{H}_{j,j',k}(t)^2 \,\mathrm{d}N_j(t)}{x} \vee 2\big) + 2\log\log(4\sup_{0\le t\le T}|\mathbb{H}_{j,j',k}(t)| \vee 2)$. Once again, this is natural: the variance term $\int_0^T \mathbb{H}_{j,j',k}(t)^2 \,\mathrm{d}N_j(t)$ is, roughly, an estimation of the variance of the self-excitements between coordinates j and j' at time scale k. The term $\mathbb{L}_{j,j',k}(T)$ is a technical term that can be neglected in practice.

These weights are actually quite natural: the terms $\lambda_{\max}(\widehat{\boldsymbol{V}}(T))$ and $\int_0^T \mathbb{H}_{j,j',k}(t)^2 \,\mathrm{d}N_j(t)$ correspond to estimations of the noise variance, that are the L^2 terms appearing in the empirical Bernstein's inequalities given in Section 3.3. The terms $\sup_{0\le t\le T}\|\boldsymbol{H}(t)\|_{\infty,2}$ and $\sup_{0\le t\le T}|\mathbb{H}_{j,j',k}(t)|$ correspond to the L^∞ terms from these Bernstein's inequalities. Once again, these data-driven weights lead to a sharp tuning of the penalizations, as illustrated numerically in Section 6 below.

5 A sharp oracle inequality

Recall that the inner product $\langle \lambda_1, \lambda_2 \rangle_T$ is given by (II.5) and recall that $\|\cdot\|_T$ stands for the corresponding norm. Theorem 4 below is a sharp oracle inequality on the

prediction error measured by $\|\lambda_{\hat{\theta}} - \lambda\|_T^2$. For the proof of oracle inequalities with a fast rate, one needs a restricted eigenvalue condition on the Gram matrix of the problem [BRT09, Koll1]. One of the weakest assumptions considered in literature is the Restricted Eigenvalue (RE) condition. In our setting, a natural RE assumption is given in Definition 1 below. First, we need to introduce some simple notations and definitions.

Some notations and definitions. If a, b (resp. $\boldsymbol{A}, \boldsymbol{B}$ and $\mathbb{A}, \mathbb{B}$) are vectors (resp. matrices and tensors) of the same size, we always denote by $\langle a, b\rangle$ (resp. $\langle \boldsymbol{A}, \boldsymbol{B}\rangle$ and $\langle \mathbb{A}, \mathbb{B}\rangle$) their inner products. For matrices this can be written as $\langle \boldsymbol{A}, \boldsymbol{B}\rangle = \sum_{i,j} \boldsymbol{A}_{i,j}\boldsymbol{B}_{i,j} = \mathrm{tr}(\boldsymbol{A}^\top \boldsymbol{B})$, where tr stands for the trace, while for (say, three dimensional) tensors we write similarly $\langle \mathbb{A}, \mathbb{B}\rangle = \sum_{i,j,k} \mathbb{A}_{i,j,k}\mathbb{B}_{i,j,k}$. We define the Euclidean norm (Frobenius) for tensors and matrices simply as $\|\boldsymbol{A}\|_F = \sqrt{\langle \boldsymbol{A}, \boldsymbol{A}\rangle}$ and $\|\mathbb{A}\|_F = \sqrt{\langle \mathbb{A}, \mathbb{A}\rangle}$. If $\boldsymbol{W}$ (resp. $\mathbb{W}$) is a matrix (resp. tensor) with positive entries, we introduce the weighted entrywise ℓ_1-norm given by $\|\boldsymbol{A}\|_{1,\boldsymbol{W}} = \langle \boldsymbol{W}, |\boldsymbol{A}|\rangle$, (resp. $\|\mathbb{A}\|_{1,\mathbb{W}} = \langle \mathbb{W}, |\mathbb{A}|\rangle$) where $|\boldsymbol{A}|$ (resp. $|\mathbb{A}|$) contains the absolute values of the entries of $\boldsymbol{A}$ (resp. $\mathbb{A}$). If A is a vector, matrix or tensor then $\|A\|_0$ is the number of non-zero entries of A, while $\mathrm{supp}(A)$ stands for the support of A (indices of non-zero entries). For another vector, matrix or tensor A' with the same shape, the notation $[A']_{\mathrm{supp}(A)}$ stands for the vector, matrix or tensor with the same coordinates as A' where we put 0 at indices outside of $\mathrm{supp}(A)$. We also use the notation $u \vee v = \max(u, v)$ for $a, b \in \mathbb{R}$.

If $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ is the SVD of a $m \times n$ matrix $\boldsymbol{A}$, with the columns u_j of $\boldsymbol{U}$ and v_k of $\boldsymbol{V}$ being, respectively, the orthonormal left and right singular vectors of $\boldsymbol{A}$, the projection matrix onto the space spanned by the columns (resp. rows) of $\boldsymbol{A}$ is given by $\boldsymbol{P}_{\boldsymbol{U}} = \boldsymbol{U}\boldsymbol{U}^\top$ (resp. $\boldsymbol{P}_{\boldsymbol{V}} = \boldsymbol{V}\boldsymbol{V}^\top$). The operator $\mathscr{P}_{\boldsymbol{A}} : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ given by $\mathscr{P}_{\boldsymbol{A}}(\boldsymbol{B}) = \boldsymbol{P}_{\boldsymbol{U}}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{P}_{\boldsymbol{V}} - \boldsymbol{P}_{\boldsymbol{U}}\boldsymbol{B}\boldsymbol{P}_{\boldsymbol{V}}$ is the projector onto the linear space spanned by the matrices $u_j x^\top$ and $y v_k^\top$ for all $1 \le j, k \le \mathrm{rank}(\boldsymbol{A})$ and $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$. The projector onto the orthogonal space is given by $\mathscr{P}^\perp_{\boldsymbol{A}}(\boldsymbol{B}) = (\boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{U}})\boldsymbol{B}(\boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{V}})$.

Definition 1. *Fix $\theta = (\mu, \mathbb{A})$ where $\mu \in \mathbb{R}^d$ and $\mathbb{A} \in \mathbb{R}_+^{d\times d\times K}$ and define $\boldsymbol{A} = \mathrm{hstack}(\mathbb{A})$. We define the constant $\kappa(\theta) \in (0, +\infty]$ such that, for any $\theta' = (\mu', \mathbb{A}')$ and $\boldsymbol{A}' = \mathrm{hstack}(\mathbb{A}')$ satisfying*

$$\begin{aligned}\frac{1}{3}\|(\mu')_{\mathrm{supp}(\mu)^\perp}\|_{1,\hat{w}} + \frac{1}{2}\|(\mathbb{A}')_{\mathrm{supp}(\mathbb{A})^\perp}\|_{1,\hat{\mathbb{W}}} + \frac{1}{2}\hat{\tau}\|\mathscr{P}^\perp_{\boldsymbol{A}}(\boldsymbol{A}')\|_* \\ \le \frac{5}{3}\|(\mu')_{\mathrm{supp}(\mu)}\|_{1,\hat{w}} + \frac{3}{2}\|(\mathbb{A}')_{\mathrm{supp}(\mathbb{A})}\|_{1,\hat{\mathbb{W}}} + \frac{3}{2}\hat{\tau}\|\mathscr{P}_{\boldsymbol{A}}(\boldsymbol{A}')\|_*,\end{aligned}$$

we have

$$\|(\mu')_{\mathrm{supp}(\mu)}\|_2 \vee \|(\mathbb{A}')_{\mathrm{supp}(\mathbb{A})}\|_F \vee \|\mathscr{P}_{\boldsymbol{A}}(\boldsymbol{A}')\|_F \le \kappa(\theta)\|\lambda_{\theta'}\|_T.$$

The constant $1/\kappa(\theta)$ is a restricted eigenvalue depending on the "support" of θ, which is naturally associated with the problem considered here. Roughly, it requires

that for any parameter θ' that has a support close to the one of θ (measured by domination of the ℓ_1 norms outside the support of θ by the ℓ_1 norm inside it), we have that the L^2 norm of the intensity given by $\|\lambda_{\theta'}\|_T$ can be compared with the L^2 norm of θ' in the support of θ. Note that for a given θ, we simply allow $\kappa(\theta) = +\infty$, so the restricted eigenvalue is zero, whenever the inequality is not met (which makes in such a case the statement of Theorem 4 trivial).

Theorem 4. *Fix $x > 0$, and let $\hat{\theta}$ be given by* (II.12) *and* (II.16) *with tuning parameters given by* (II.17), (II.18) *and* (II.19). *Then, the inequality*

$$\|\lambda_{\hat{\theta}} - \lambda\|_T^2 \leq \inf_{\theta=(\mu,\mathbb{A})} \Big\{ \|\lambda_\theta - \lambda\|_T^2 + 1.25\kappa(\theta)^2 \Big(\|(\hat{w})_{\mathrm{supp}(\mu)}\|_2^2 + \|(\hat{\mathbb{W}})_{\mathrm{supp}(\mathbb{A})}\|_F^2 + \hat{\tau}^2 \,\mathrm{rank}(\mathrm{hstack}(\mathbb{A})) \Big) \Big\}$$

holds with a probability larger than $1 - 70.35e^{-x}$.

The proof of Theorem 4 is given in Section 8.5 below. Note that no assumption is required on the ground truth intensity λ of the multivariate counting process N in Theorem 4. Moreover, if one forgets in Section 4 about the negligible terms $\ell_\tau(T), \ell_j(T)$ and $\mathbb{L}_{j,j',k}(T)$ and if one keeps only the dominating L^2 terms in $O(1/T)$ (while L^∞ terms are $O(1/T^2)$ in the large T regime), we obtain upper bounds, up to numerical constants (denoted $\lesssim$), for the terms involved in Theorem 5:

$$\|(\hat{w})_{\mathrm{supp}(\mu)}\|_2^2 \lesssim \|\mu\|_0 \max_{j \in \mathrm{supp}(\mu)} \frac{\frac{1}{T} N_j(T)(x + \log d)}{T},$$

where $\|\mu\|_0$ stands for the sparsity of μ,

$$\|(\hat{\mathbb{W}})_{\mathrm{supp}(\mathbb{A})}\|_F^2 \lesssim \|\mathbb{A}\|_0 \max_{(j,j',k) \in \mathrm{supp}(\mathbb{A})} \frac{\frac{1}{T} \int_0^T \mathbb{H}_{j,j',k}(t)^2 \,\mathrm{d}N_j(t)(x + \log(Kd^2))}{T},$$

where $\|\mathbb{A}\|_0$ stands for the sparsity of $\mathbb{A}$, and finally

$$\hat{\tau}^2 \lesssim \mathrm{rank}(\mathrm{hstack}(\mathbb{A})) \frac{\frac{1}{T} \lambda_{\max}(\widehat{\boldsymbol{V}}(T))(x + \log(2d))}{T}.$$

Hence, Theorem 4 proves that $\hat{\theta}$ achieves an optimal trade-off between approximation and complexity, where the complexity is, roughly, measured by

$$\frac{\|\mu\|_0 (x + \log d)}{T} \max_j \frac{N_j(T)}{T} + \frac{\|\mathbb{A}\|_0 (x + \log(Kd^2))}{T} \max_{j,j',k} \frac{1}{T} \int_0^T \mathbb{H}_{j,j',k}(t)^2 \,\mathrm{d}N_j(t) + \frac{\mathrm{rank}(\mathrm{hstack}(\mathbb{A}))(x + \log(2d))}{T} \frac{1}{T} \lambda_{\max}(\widehat{\boldsymbol{V}}(T)).$$

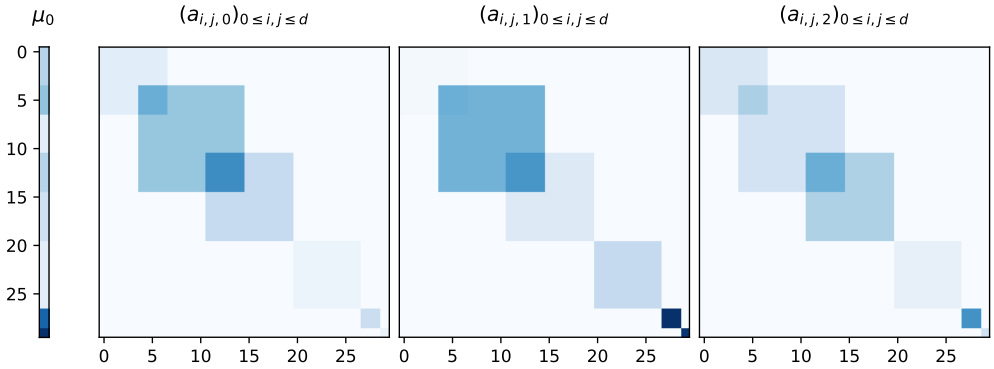

Figure II.2: Ground truth μ vector and $\mathbb{A}$ matrix in dimension 30 and number of ticks per node after a simulation over an interval of time of 20000

Note that typically $K \le d$ so that $\log(Kd^2) \le 3 \log d$, which means that $\log(Kd^2)$ scales as $\log d$. The complexity term depends on both the sparsity of $\mathbb{A}$ and the rank of hstack($\mathbb{A}$). The rate of convergence has the "expected" shape $(\log d)/T$, recalling that T is the length of the observation interval of the process, and these terms are balanced by the empirical variance terms coming out of the new concentration results given in Section 3.3 above.

6 Numerical experiments

In this Section we conduct experiments on synthetic datasets to evaluate the performance of our method, based on the proposed data-driven weighting of the penalizations, compared to non-weighted penalizations [ZZS13a]. Throughout this Section, we consider the most widely used sum of exponentials kernel, defined in Equation (II.3).

Simulation setting. We generate Hawkes processes using Ogata's thinning algorithm [Oga81] with $d = 30$ nodes. Baseline intensities μ_j are constant on blocks, we use $K = 3$ basis kernels $h_{j,j',k}(t) = \alpha_k e^{-\alpha_k t}$ with $\alpha_1 = 0.5$, $\alpha_1 = 2$ and $\alpha_3 = 5$. The slices $\mathbb{A}_{\bullet,\bullet,1}$, $\mathbb{A}_{\bullet,\bullet,2}$ and $\mathbb{A}_{\bullet,\bullet,3}$ of the adjacency tensor $\mathbb{A}$ contains square overlapping boxes, as illustrated in Figure II.2. These boxes correspond to the overlapping communities reacting at different time scales. Each box is filled with constant values and the rest of the matrix contains zeros. The tensor $\mathbb{A}$ is rescaled so that the operator norm of the matrix $\sum_{k=1}^{3} \mathbb{A}_{\bullet,\bullet,k}$ is equal to 0.8, guaranteeing to obtain a stationary process. For each simulated data, we increase the length of the time interval $T = 5000, 7000, 10000, 15000, 20000$, and fit each time the procedures. An overall averaging of the results is computed on 100 separate simulations.

Procedures. We consider a procedure based on the minimization of the least-squares functional (II.4). This objective is convex, with a goodness-of-fit term which is gradient-Lipschitz: we use first-order optimization algorithms, based on proximal gradient descent. Namely, we use FISTA [BT09] for problems with a single penalization on $\mathbb{A}$ (ℓ_1-norm) and GFB (generalized forward backward) [PLR+99] for mixed ℓ_1 penalization of $\mathbb{A}$ and trace-norm penalization of hstack($\mathbb{A}$). For both procedures we choose a fixed gradient step equal to $1/L$ where L is the Lipschitz constant of the loss, namely the largest singular value of the Hessian (which is constant for this least-squares functional). We limit our algorithms to 25,000 iterations and stop when the objective relative decrease is less than 10^{-10} for FISTA and 10^{-7} for GFB. We only penalize $\mathbb{A}$ and consider the following procedures:

- L1: non-weighted L1 penalization;
- wL1: weighted L1 penalization;
- L1Nuclear: non-weighted L1 penalization and trace-norm penalization;
- wL1Nuclear: weighted L1 penalization and trace-norm penalization.

Note that L1Nuclear is the same as the procedure considered in [ZZS13a], however, we use a different optimization algorithm, based on an proximal gradient descent (a first-order method, which is typically faster than an algorithm based on ADMM, as proposed in [ZZS13a]). The data-driven weights used in our procedures are the ones derived from our analysis, see (II.17) and (II.19), where we simply put $x = \log T$. For each metric, we tune the constant in front the ℓ_1 penalization, and the constant in front of the trace-norm penalization in order to obtain the best possible metrics for each procedure, on average over all separate simulations. Namely, there is no test set, we simply display the best metrics obtained by each procedure for a fair comparison. All experiments are done using our `tick` library for `Python3`, see Chapter V.

Metrics. The following metrics are considered in order to assess the procedures.

- Estimation error: the relative ℓ_2 estimation error of $\mathbb{A}$, given by $\|\hat{\mathbb{A}} - \mathbb{A}\|_2^2 / \|\mathbb{A}\|_2^2$
- AUC: we compute the AUC (area under the ROC curve) between the binarized ground truth matrix $\mathbb{A}$ and the solution $\hat{\mathbb{A}}$ with entries scaled in $[0, 1]$. This allows us to quantify the ability of the procedure to detect the support of the connectivity structure between nodes.
- Kendall: we compute Kendall's tau-b between all entries of the ground truth matrix $\mathbb{A}$ and the solution $\hat{\mathbb{A}}$. This correlation coefficient takes value between -1 and 1 and compare the number of concordant and discordant pairs. This

allows us to quantify the ability of the procedure to rank correctly the intensity of the connectivity between nodes.

Results. In Figure II.3 we observe, on an instance of the problem, the strong improvements of wL1 and wL1Nuclear over L1 and L1Nuclear respectively. We observe in particular that a sharp tuning of the penalizations, using data-driven weights, leads to a much smaller number of false positives outside the node communities (better viewed on a computer). In Figure II.4, we compare all the procedures in terms of estimation error, AUC and Kendall coefficient and confirm the fact that weighted penalizations systematically lead to an improvement, both over unweighted L1 and L1Nuclear.

A comparison of the least-squares and likelihood functionals. This paper considers, mostly for theoretical reasons, least-squares as a goodness-of-fit for the Hawkes process. However, estimation in this model is usually achieved by minimizing the goodness-of-fit given by the negative log-likelihood. In what follows, we provide some numerical insights in order to compare objectively both approaches.

First, one can precompute for both functionals some weights in order to accelerate future gradient and value computations. In both cases, the precomputations have similar complexities, unless the number of kernels K is large (see Table II.1 below). However, given such precomputations, a remarkable property of the least-squares versus the log likelihood is that value and gradient computation is independent of the total number of observed events (denoted n): complexity is $O(K^2d^3)$ for least-squares, while it is $O(nKd)$ for log likelihood, which means that such computations for least-squares is orders of magnitude faster, since typically $n \gg Kd^2$ (one needs to observe hundreds of events for each pair of nodes for a good inference of the model). For instance, experiments used to produce Figures II.3 and II.4 for $T = 20,000$ use about $n \approx 500,000$ events, and $d = 30, K = 3$. The complexity of each operation is described in Table II.1 below and a numerical illustration of this complexity is displayed in Figure II.5, which confirms that computations with least-squares are orders of magnitude faster than with log-likelihood. These complexities are detailed in Chapter I.

Another important point is related to smoothness properties: the negative log-likelihood does not satisfy the gradient-Lipschitz assumption, while this property is required by most first order optimization algorithms to obtain convergence guarantees and an easy tuning of the step-size used in gradient descent. Therefore, for the negative log-likelihood, convergence can be very unstable, while on the contrary, least-squares is gradient-Lipschitz and is easy to optimize since it is a quadratic function. Note that Chapter IV proposes an alternative approach based on duality, in particular for the negative log-likelihood of the Hawkes process. Herein one can

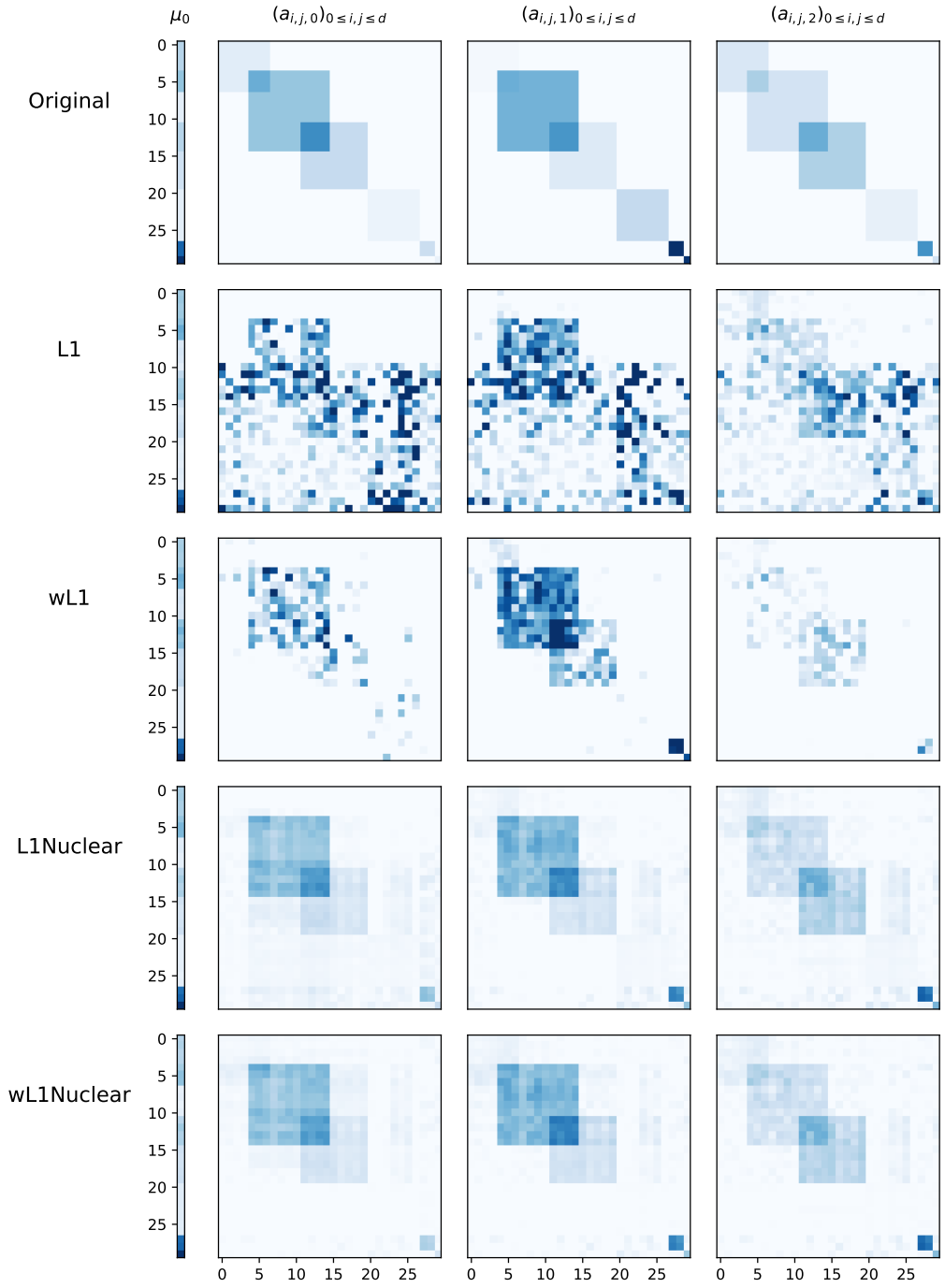

Figure II.3: Ground truth tensor $\mathbb{A}$ and recovered tensors using all procedures. We observe that wL1 and wL1Nuclear leads to a much better support recovery, as we observe less false positives outside of the node communities.

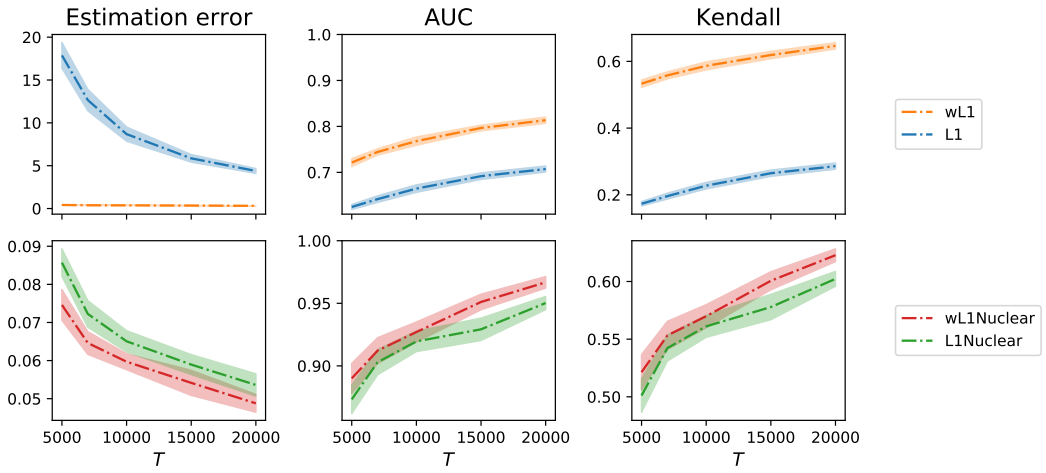

Figure II.4: Average metrics achieved by all procedures (and 95% confidence bands) with increasing observation length T over repeated simulations. Weighted penalizations systematically lead to strong improvements, both for L1 and L1 + Nuclear penalization.

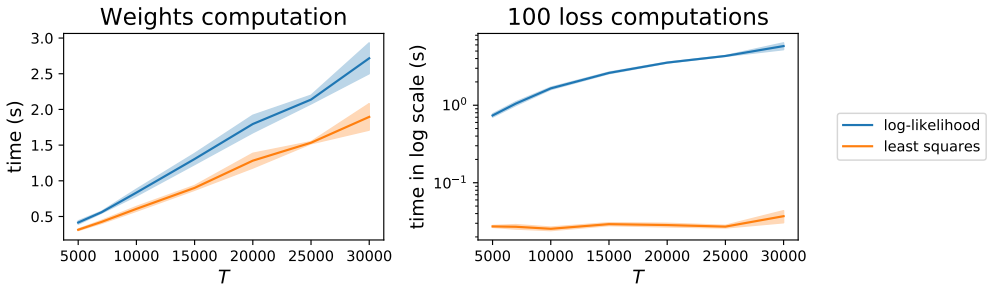

Figure II.5: Average time needed for weights (left) and value computation (right) (and 95% confidence bands) for least squares and log-likelihood with precomputations, over repeated simulations. We observe that value computations are order of magnitude faster for least-squares (y-scale is logarithmic on the right hand side) and constant with an increasing observation length, while it is strongly increasing for the log-likelihood.

	pre-computation	memory	value	gradient
Least squares	$O(nK^2d)$	$O(K^2d^3)$	$O(K^2d^3)$	$O(K^2d^3)$
Likelihood	$O(nKd)$	$O(nKd)$	$O(nKd)$	$O(nKd)$

Table II.1: From left to right: Weights precomputation complexity, memory storage, value and gradient complexity for both functionals. Note that for least-squares, the complexity of the value and the gradient with precomputed weights is independent on the number of events n.

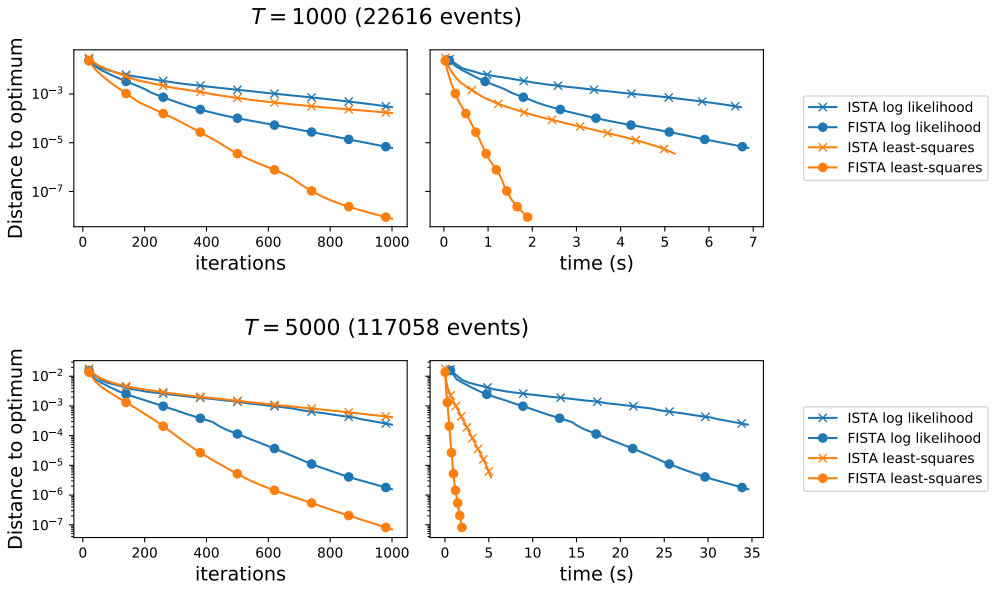

Figure II.6: Convergence speed of least squares and likelihood losses with ISTA and FISTA optimization algorithms on two simulations of a Hawkes process with parameters from Figure II.2 with observation length $T = 1000$ (top) and $T = 5000$ (bottom). Once again, we observe that the computations are much faster with least-squares, in particular with a large observation length.

observe the strong instability of standard first order algorithms (such as the one considered here) for the negative log-likelihood.

In Figure II.6 below, we compare the performances of ISTA and FISTA with linesearch for automatic step-size tuning, both for least-squares and negative log-likelihood. This figure confirms that the number of iterations required for least-squares is much smaller than for the negative log-likelihood. This gap is even stronger if we look at the computation times, since each iteration is computationally faster with least squares, and even more so when the observation length increases.

In this Section, we compared least-squares and log-likelihood for the Hawkes

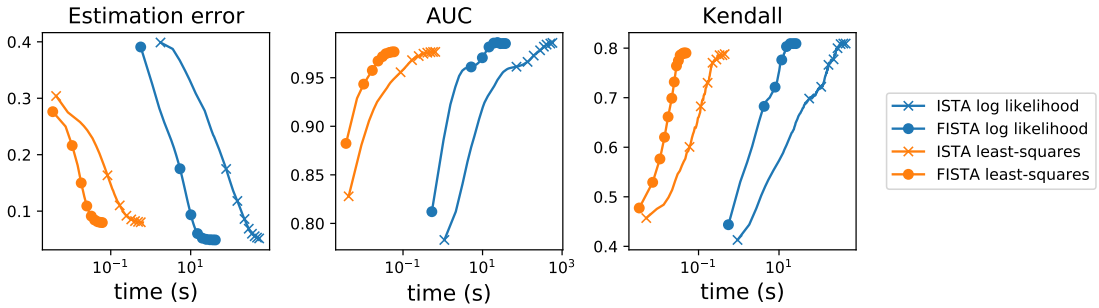

Figure II.7: Metrics achieved by least squares and log-likelihood estimators after precomputations. We observe that log-likelihood achieves a slightly better AUC and Estimation Error, but with larger computational cost (x-axis are on a logarithmic scale).

process through a computational perspective only, and concluded that least-squares is typically order of magnitude faster. Now, let us compare the statistical performances of both approaches on the same simulation setting as before, with $T = 20,000$, using the metrics defined above, namely Estimation Error, AUC and Kendall. We simply use for this L1 penalization on $\mathbb{A}$, with a strength parameter tuned for each metric and for each goodness-of-fit. In Figure II.7, we observe that both functionals roughly achieve the same performance measured by the Kendall coefficient, but that the negative log-likelihood achieves a slightly better AUC and estimation error than least-squares, at a larger computational cost. The slightly better statistical performance of maximum likelihood is not surprising, since vanilla maximum likelihood is known to be statistically efficient asymptotically for Hawkes processes, see [Oga78], while up to our knowledge, vanilla least-squares estimator is not. This leads to the conclusion that least squares are a very good alternative to maximum likelihood when dealing with a large number of events: statistical accuracy is only slightly deteriorated, but the computational cost is order of magnitudes smaller, and convergence is much more stable.

7 Conclusion

In this paper we proposed a careful analysis of the generalization error of the multivariate Hawkes process. Our theoretical analysis required a new concentration inequality for matrix-martingales in continuous time, with an observable variance term, which is a result of independent interest. This analysis led to a new data-driven tuning of sparsity-inducing penalizations, that we assessed on a numerical example. Future works will focus on other theoretical results for non-convex matrix factorization tech-

niques applied to this problem.

Acknowledgments

This research benefited from the support of the "Chair Markets in Transition", under the aegis of "Louis Bachelier Finance and Sustainable Growth" laboratory, a joint initiative of École Polytechnique, Université d'Évry Val d'Essonne and Fédération Bancaire Francaise.

8 Proofs

This Section contains the proofs of all the results given in the paper. First, we prove the statements concerned with deviation inequalities, namely Theorems 1, 2, Proposition 1 and Theorem 3. Then, we give the proof of Theorem 4, concerning the oracle inequality for the procedure.

8.1 Proof of Theorem 1

In [BGM18], a deviation inequality is proven in a slightly more general setting than the one considered in this paper. There are mainly two differences.

- This paper considers only counting processes with uniform jumps of size 1 whereas in [BGM18], jump sizes are controlled by a predictable process $\boldsymbol{J}$. Therefore, it suffices to set $\boldsymbol{J} = \mathbf{1}$ and $\boldsymbol{C}_s = \mathbf{1}$ in Equations (2) and (3) of [BGM18], where $\mathbf{1}$ stands for the all-ones matrices with relevant shapes.

- In [BGM18], the deviation inequality is proved in a general context where no symmetry is assumed on $\mathbb{T}_s$. It forces to consider a symmetric version of $\boldsymbol{W}_{\mathbb{T}}(s)$ as in Eq. (II.7) increasing the dimension of the working space by a factor of 2, which leads to less precise deviation inequality. In this paper we consider both cases, symmetric and non symmetric, in order to obtain slightly better constants (see the definition of $K_{m,n}$).

With those two differences in mind, following carefully the proof of the concentration inequality in [BGM18] (see the beginning of Appendix B.1 herein) one gets

$$\mathbb{P}\Big[\frac{\lambda_{\max}(\mathscr{S}(\boldsymbol{Z}_t))}{b} \geq \frac{1}{\xi}\lambda_{\max}\Big(\int_0^t \frac{\phi\big(\xi J_{\max}\|\boldsymbol{C}_s\|_\infty \max(\|\mathbb{T}_s\|_{\mathrm{op};\infty}, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty}) b^{-1}\big)}{J_{\max}^2\|\boldsymbol{C}_s\|_\infty^2 \max(\|\mathbb{T}_s\|_{\mathrm{op};\infty}^2, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty}^2)} \boldsymbol{W}_s \,\mathrm{d}s\Big) + \frac{x}{\xi}, \\ b_{\mathbb{T}}(t) \leq b\Big] \leq (m+n)e^{-x},$$

where $\xi \in (0,3)$ and $\lambda_{\max}(\mathscr{S}(\boldsymbol{Z}_t)) = \|\boldsymbol{Z}\|_{\mathrm{op}}$ (see the beginning of Appendix B.1 in [BGM18]). Setting $\boldsymbol{J} = \mathbf{1}$, $\boldsymbol{C} = \mathbf{1}$ and taking care of the symmetric case at the same time as the non symmetric one, one gets:

$$\mathbb{P}\Big[\frac{\|\boldsymbol{Z}_t\|_{\mathrm{op}}}{b} \geq \frac{1}{\xi}\lambda_{\max}\Big(\int_0^t \frac{\phi\big(\xi \max(\|\mathbb{T}_s\|_{\mathrm{op};\infty}, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty})b^{-1}\big)}{\max(\|\mathbb{T}_s\|_{\mathrm{op};\infty}^2, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty}^2)} \boldsymbol{W}_s \,\mathrm{d}s\Big) + \frac{x}{\xi}, \\ b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x},$$

using the definitions $K_{m,n}$ and $\boldsymbol{W}_s$ introduced previously (depending on the symmetric properties of the tensor $\mathbb{T}_s$). Let us note that on $\{b_{\mathbb{T}}(t) \leq b\}$ one has $\max(\|\mathbb{T}_s\|_{\mathrm{op};\infty}, \|\mathbb{T}_s^\top\|_{\mathrm{op};\infty})b^{-1} \leq 1$ for any $s \in [0,t]$. Thus, since $\phi(xh) \leq h^2\phi(x)$ for any $h \in [0,1]$ and $x > 0$, one gets

$$\mathbb{P}\Big[\frac{\|\boldsymbol{Z}_t\|_{\mathrm{op}}}{b} \geq \frac{\phi(\xi)}{\xi b^2}\lambda_{\max}\Big(\int_0^t \boldsymbol{W}_s \,\mathrm{d}s\Big) + \frac{x}{\xi}\ , \quad b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x}$$

and finally

$$\mathbb{P}\Big[\|\boldsymbol{Z}_t\|_{\mathrm{op}} \geq \frac{\phi(\xi)}{\xi b}\lambda_{\max}(\boldsymbol{V}_t) + \frac{xb}{\xi}\ , \quad b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x}$$

which proves the first part of the Theorem. The second part (i.e., Inequality (II.9)) can be obtained following some standard tricks (see e.g. [Mas07]):

(i) on $(0,3)$, $\phi(\xi) \leq \frac{\xi^2}{2(1-\xi/3)}$ and

(ii) $\min_{\xi \in (0,1/c)}\big(\frac{a\xi}{1-c\xi} + \frac{x}{\xi}\big) = 2\sqrt{ax} + cx$ for any $a, c, x > 0$.

Thus applying (i) leads to

$$\mathbb{P}\Big[\|\boldsymbol{Z}_t\|_{\mathrm{op}} \geq \frac{\xi}{2b(1-\xi/3)}\lambda_{\max}(\boldsymbol{V}_t) + \frac{xb}{\xi}\ , \quad b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x}$$

or equivalently

$$\mathbb{P}\Big[\|\boldsymbol{Z}_t\|_{\mathrm{op}} \geq \frac{\xi}{2b(1-\xi/3)}v + \frac{xb}{\xi}\ , \quad \lambda_{\max}(\boldsymbol{V}_t) \leq v,\ b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x}.$$

Then optimizing on ξ using (ii) with $c = 1/3$ and $a = v/2b^2$, one gets

$$\mathbb{P}\Big[\|\boldsymbol{Z}_t\|_{\mathrm{op}} \geq \sqrt{2vx} + \frac{xb}{3}\ , \quad \lambda_{\max}(\boldsymbol{V}_t) \leq v,\ b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x}$$

which concludes the proof of Theorem 1. ■

8.2 Proof of Proposition 1

This Proposition provides a deviation between $\lambda_{\max}(\boldsymbol{V}(t))$ and $\lambda_{\max}(\widehat{\boldsymbol{V}}(t))$. Let us notice that it is a generalization to arbitrary matrices of dimensions $m \times n$ of an analog inequality originally proven by Hansen et al. [HRBR15] for scalar martingales (i.e., in dimension 1). The proof below follows the same lines as these authors. The proof is based on the observation that the difference $\boldsymbol{V}_{\mathbb{T}}(t) - \widehat{\boldsymbol{V}}_{\mathbb{T}}(t)$ can be written as a martingale $\boldsymbol{Z}_{\mathbb{H}}(t)$

$$\boldsymbol{V}_{\mathbb{T}}(t) - \widehat{\boldsymbol{V}}_{\mathbb{T}}(t) = \boldsymbol{Z}_{\mathbb{H}}(t) = \int_0^t \mathbb{H}_s \circ \mathrm{d}\boldsymbol{M}_s,$$

where

$$\mathbb{H}_s = \mathbb{T}_s^2$$

when $\mathbb{T}_s$ is symmetric, while

$$\mathbb{H}_s = \begin{bmatrix} \mathbb{T}_s \mathbb{T}_s^\top & \mathbf{0} \\ \mathbf{0} & \mathbb{T}_s^\top \mathbb{T}_s \end{bmatrix}$$

if $\mathbb{T}_s$ is not symmetric. Then applying Eq. (II.8) of Theorem 1 to the martingale $\boldsymbol{Z}_{\mathbb{H}}(t)$ (we are in the symmetric case of the Theorem since $\mathbb{H}_s^\top = \mathbb{H}_s$), one gets

$$\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{H}}(t)\|_{\mathrm{op}} \geq \frac{\phi(\xi)}{\xi b} \lambda_{\max}\big(\boldsymbol{V}_{\mathbb{H}}(t)\big) + \frac{xb}{\xi}, \quad b_{\mathbb{H}}(t) \leq b \Big] \leq K_{m,n} e^{-x},$$

with

$$\boldsymbol{V}_{\mathbb{H}}(t) = \int_0^t \mathbb{H}_s^2 \circ \boldsymbol{\lambda}_s \, \mathrm{d}s \,. \tag{II.20}$$

Since

$$\|\boldsymbol{Z}_{\mathbb{H}}(t)\|_{\mathrm{op}} \geq \lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) - \lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)),$$

we have

$$\mathbb{P}\Big[\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) \geq \lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) + \frac{\phi(\xi)}{\xi b} \lambda_{\max}\big(\boldsymbol{V}_{\mathbb{H}}(t)\big) + \frac{xb}{\xi}, \quad b_{\mathbb{H}}(t) \leq b \Big] \leq K_{m,n} e^{-x}, \tag{II.21}$$

One can first notice that, from the definitions of $\mathbb{H}$ and $b_{\mathbb{T}}(t)$, one has $b_{\mathbb{H}}(t) \leq b_{\mathbb{T}}^2(t)$. Moreover, since

$$\mathbb{T}_s \mathbb{T}_s^\top \preccurlyeq b_{\mathbb{T}}^2(s) \boldsymbol{I}_m \quad \text{and} \quad \mathbb{T}_s^\top \mathbb{T}_s \preccurlyeq b_{\mathbb{T}}^2(s) \boldsymbol{I}_n$$

for all s, we have from Eq. (II.20),

$$\boldsymbol{V}_{\mathbb{H}}(t) \preccurlyeq b_{\mathbb{T}}^2(t) \boldsymbol{V}_{\mathbb{T}}(t)$$

and therefore

$$\lambda_{\max}(\boldsymbol{V}_{\mathbb{H}}(t)) \leq b_{\mathbb{T}}^2(t)\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)).$$

Inequality (II.21) then gives:

$$\mathbb{P}\Big[\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) \geq \lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) + \frac{\phi(\xi)}{\xi}\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) + \frac{xb^2}{\xi}, \quad b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x},$$

and thus

$$\mathbb{P}\Big[\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) \geq \frac{\xi\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))}{\xi - \phi(\xi)} + \frac{xb^2}{\xi - \phi(\xi)}, \quad b_{\mathbb{T}}(t) \leq b\Big] \leq K_{m,n}e^{-x},$$

which proves the first inequality stated in Proposition 1. Now, an easy computation proves that the choice $\xi = -W_{-1}(-\frac{2}{3}e^{-2/3}) - 2/3 \approx 0.762$ provides the second desired inequality. □

8.3 Proof of Theorem 2

Introduce the set

$$E_t = \{\lambda_{\max}(\boldsymbol{V}_{\mathbb{T}}(t)) \leq 2\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) + 2.62b^2x\}.$$

We know from Proposition 1 that $\mathbb{P}[E_t^{\complement}, b_{\mathbb{T}}(t) \leq b] \leq K_{m,n}e^{-x}$. Now, on the set

$$E_t \cap \{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \leq v\} \cap \{b_{\mathbb{T}}(t) \leq b\}$$

we have

$$\frac{\phi(\xi)}{\xi b}\lambda_{\max}(\boldsymbol{V}(t)) + \frac{xb}{\xi} \leq \frac{\phi(\xi)}{\xi b}2v + \frac{bx}{\xi} + \frac{2.62\phi(3)}{3}bx$$

for any $\xi \in (0,3)$, since $\xi \mapsto \phi(\xi)/\xi$ is increasing. Using again points (i) and (ii) from Section 8.1 proves that the minimum for $\xi \in (0,3)$ of the right hand size of this last inequality is equal to

$$2\sqrt{vx} + \frac{2.62\phi(3)+1}{3}xb \leq 2\sqrt{vx} + cxb$$

with $c = 14.39$. Now, the conclusion easily follows from the following decomposition:

$$\begin{aligned}
&\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \geq 2\sqrt{vx} + cbx,\ \lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \leq v,\ b_{\mathbb{T}}(t) \leq b\Big] \\
&\quad \leq \mathbb{P}[E_t^{\complement}, b_{\mathbb{T}}(t) \leq b] + \mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \geq 2\sqrt{vx} + cbx,\ E_t,\ \lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \leq v,\ b_{\mathbb{T}}(t) \leq b\Big] \\
&\quad \leq K_{m,n}e^{-x} + \mathbb{P}\Big[\|\boldsymbol{Z}_t\|_{\mathrm{op}} \geq \frac{\xi}{2b(1-\xi/3)}\lambda_{\max}(\boldsymbol{V}_t) + \frac{xb}{\xi}\ ,\quad b_{\mathbb{T}}(t) \leq b\Big] \\
&\quad \leq 2K_{m,n}e^{-x},
\end{aligned}$$

where we used Equation (II.8) from Theorem 1 in the last inequality. ■

8.4 Proof of Theorem 3

In order to prove this theorem, we are going to use peeling arguments. For any $\epsilon > 0$ and $z > 0$ we define the interval

$$\mathscr{I}_{z,\varepsilon} = [z, z(1+\varepsilon)].$$

Let, $v_0, b_0, \epsilon > 0$ and let us define $v_j = v_0(1+\varepsilon)^j$, $b_j = b_0(1+\varepsilon)^j$. Let us define also the events

$$V_{-1} = \{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \le v_0\}, \quad B_{-1} = \{b_{\mathbb{T}}(t) \le b_0\},$$

and

$$V_j = \{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \in \mathscr{I}_{v_j,\varepsilon}\}, \quad B_j = \{b_{\mathbb{T}}(t) \in \mathscr{I}_{b_j,\varepsilon}\}$$

for any $j \in \mathbb{N}$. We set $v_0 = w_0 x$, then, from Equation (II.10), one gets successively

$$\begin{aligned}
\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge x\big(2\sqrt{w_0} + cb_0\big), V_{-1} \cap B_{-1}\Big] &\le 2K_{m,n}e^{-x} \\
\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge x\big(2\sqrt{w_0} + c(1+\varepsilon)b_{\mathbb{T}}(t)\big), V_{-1} \cap B_j\Big] &\le 2K_{m,n}e^{-x} \\
\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(1+\varepsilon)x} + cxb_0, V_i \cap B_{-1}\Big] &\le 2K_{m,n}e^{-x} \\
\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(1+\varepsilon)x} + c(1+\varepsilon)xb_{\mathbb{T}}(t), V_i \cap B_j\Big] &\le 2K_{m,n}e^{-x}
\end{aligned}$$

for all $i, j \ge 0$. If one denotes $A = 2\sqrt{w_0}/c + b_0$, previous inequalities entail, for any $i, j \ge -1$:

$$\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(1+\varepsilon)x} + c(1+\varepsilon)(A + b_{\mathbb{T}}(t))x, V_i \cap B_j\Big] \le 2K_{m,n}e^{-x}. \tag{II.22}$$

Let $\alpha > 0$ and define

$$\ell_x(t) = \alpha \log\Big(\log\Big(\frac{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))}{w_0 x}(1+\epsilon)^2 \vee (1+\epsilon)\Big)\Big) + \alpha \log\Big(\log\Big(\frac{b_{\mathbb{T}}(t)}{b_0}(1+\epsilon)^2 \vee (1+\epsilon)\Big)\Big).$$

Since, $\forall i, j \ge -1$, $\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)) \ge x w_0 (1+\varepsilon)^i (1-\delta_{-1,i})$ and $b_{\mathbb{T}}(t) \ge b_0(1+\varepsilon)^j(1-\delta_{-1,j})$ on $V_i \cap B_j$, then one has

$$\ell_x(t) \ge \ell_{i,j} = \log\Big((i+2)^\alpha (j+2)^\alpha (\log(1+\epsilon))^{2\alpha}\Big) \quad \text{on} \quad V_i \cap B_j$$

for any $i, j \ge -1$. Then making the change of variable $x \leftarrow x + \ell_{i,j}$ in (II.22) gives

$$\begin{aligned}
\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \ge 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(1+\varepsilon)(x+\ell_{i,j})} + c(1+\varepsilon)(A + b_{\mathbb{T}}(t))(x+\ell_{i,j}),\ V_i \cap B_j\Big] \\
\le 2K_{m,n}e^{-x}e^{-\ell_{i,j}}
\end{aligned}$$

and then

$$\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \geq 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(1+\varepsilon)(x+\ell_x(t))} + c(1+\varepsilon)(x+\ell_x(t))(A+b_{\mathbb{T}}(t)), \quad V_i \cap B_j\Big] \\ \leq 2K_{m,n}\big[\log(1+\varepsilon)\big]^{-2\alpha} e^{-x}\big[(i+2)(j+2)\big]^{-\alpha}$$

for any $i, j \geq -1$. Since the whole probability space can be partitioned as $\bigcup_{i,j\in\geq -1} V_i \cap B_j$, one has finally

$$\begin{aligned}
&\mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \geq 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(1+\varepsilon)(x+\ell_x(t))} + c(1+\varepsilon)(x+\ell_x(t))(A+b_{\mathbb{T}}(t))\Big] \\
&\quad = \sum_{i,j=-1}^{\infty} \mathbb{P}\Big[\|\boldsymbol{Z}_{\mathbb{T}}(t)\|_{\mathrm{op}} \geq 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}_{\mathbb{T}}(t))(1+\varepsilon)(x+\ell_x(t))} \\
&\qquad\qquad + c(1+\varepsilon)(x+\ell_x(t))(A+b_{\mathbb{T}}(t)), \quad V_i \cap B_j\Big] \\
&\quad \leq 2K_{m,n}\big[\log(1+\varepsilon)\big]^{-2\alpha}\Big(\sum_{i=1}^{\infty} i^{-\alpha}\Big)^2 e^{-x}.
\end{aligned}$$

Finally, choosing $\epsilon = b_0 = w_0 = 1$ and $\alpha = 2$ leads to Equation (II.11) and concludes the proof of the Theorem. ■

8.5 Proof of Theorem 4

If A, B are vectors, matrices or tensors of matching dimensions, we denote by $A \odot B$ their entrywise product (Hadamard product). We recall also that $\boldsymbol{A}_{j,\bullet}$ the j-th row of a matrix $\boldsymbol{A}$ and recall that $\|\boldsymbol{A}\|_{\infty,2} = \max_j \|\boldsymbol{A}_{j,\bullet}\|_2$. The proof is based on the proof of a sharp oracle inequality for trace norm penalization, see [KLT11] and [Koll11]. We endow the space $\mathbb{R}^d \times \mathbb{R}^{d\times d\times K}$ with the inner product

$$\langle \theta, \theta' \rangle = \langle \mu, \mu' \rangle + \langle \mathbb{A}, \mathbb{A}' \rangle,$$

where $\theta = (\mu, \mathbb{A})$ and $\theta' = (\mu', \mathbb{A}')$ with $\langle \mu, \mu' \rangle = \mu^\top \mu'$ and

$$\langle \mathbb{A}, \mathbb{A}' \rangle = \sum_{\substack{1\leq j,j'\leq d \\ 1\leq k\leq K}} \mathbb{A}_{j,j',k}\mathbb{A}'_{j,j',k}.$$

We denote for short $a_{j,j',k} = \mathbb{A}_{j,j',k}$. For any θ, one has

$$\langle \nabla R_T(\hat{\theta}), \hat{\theta} - \theta \rangle = 2\sum_{1\leq j\leq d} (\hat{\mu}_j - \mu_j)\frac{\partial R_T(\hat{\theta})}{\partial \hat{\mu}_j} + \sum_{\substack{1\leq j,j'\leq d \\ 1\leq k\leq K}} (\hat{a}_{j,j',k} - a_{j,j',k})\frac{\partial R_T(\hat{\theta})}{\partial \hat{a}_{j,j',k}}.$$

Let us recall that $\mathbb{H}_{j,j',k}(t) = \int_{(0,t)} h_{j,j',k}(t-s)\,\mathrm{d}N_{j'}(s)$. Since

$$\frac{\partial \lambda_{j,\theta}(t)}{\partial \mu_j} = 1 \quad \text{and} \quad \frac{\partial \lambda_{j,\theta}(t)}{\partial a_{j,j',k}} = \mathbb{H}_{j,j',k}(t),$$

we have that the derivatives of the empirical risk are given by

$$\frac{\partial R_T(\hat{\theta})}{\partial \mu_j} = \frac{2}{T}\Big(\int_0^T \lambda_{j,\hat{\theta}}(t)\,\mathrm{d}t - \int_0^T \mathrm{d}N_j(t)\Big)$$

and

$$\frac{\partial R_T(\hat{\theta})}{\partial a_{j,j',k}} = \frac{2}{T}\Big(\int_0^T \mathbb{H}_{j,j',k}(t)\lambda_{j,\hat{\theta}}(t)\,\mathrm{d}t - \int_0^T \mathbb{H}_{j,j',k}(t)\,\mathrm{d}N_j(t)\Big).$$

It leads to

$$\begin{aligned}\langle \nabla R_T(\hat{\theta}), \hat{\theta} - \theta\rangle &= \frac{2}{T}\sum_{j=1}^{d}\int_0^T (\lambda_{j,\hat{\theta}}(t) - \mathrm{d}N_j(t))(\hat{\mu}_j - \mu_j) \\ &\quad + \frac{2}{T}\sum_{\substack{1\le j,j'\le d\\ 1\le k\le K}}\int_0^T \mathbb{H}_{j,j',k}(t)(\lambda_{j,\hat{\theta}}(t) - \mathrm{d}N_j(t))(\hat{a}_{j,j',k} - a_{j,j',k}) \\ &= \frac{2}{T}\sum_{j=1}^{d}\int_0^T (\lambda_{j,\hat{\theta}}(t) - \lambda_{j,\theta}(t))(\lambda_{j,\hat{\theta}}(t)\,\mathrm{d}t - \mathrm{d}N_j(t)).\end{aligned}$$

Let us remind that $M_j(t) = N_j(t) - \int_0^t \lambda_j(s)\,\mathrm{d}s$ are martingales coming from the Doob-Meyer decomposition, so that $\mathrm{d}M_j(t) = \mathrm{d}N_j(t) - \lambda_j(t)\,\mathrm{d}t$. So, recalling that

$$\langle f, g\rangle_T = \frac{1}{T}\sum_{1\le j\le d}\int_{[0,T]} f_j(t)g_j(t)\,\mathrm{d}t,$$

we obtain the decomposition

$$\langle \nabla R_T(\hat{\theta}), \hat{\theta} - \theta\rangle = 2\langle \lambda_{\hat{\theta}} - \lambda_\theta, \lambda_{\hat{\theta}} - \lambda\rangle_T - \frac{2}{T}\sum_{j=1}^{d}\int_0^T (\lambda_{j,\hat{\theta}}(t) - \lambda_{j,\theta}(t))\,\mathrm{d}M_j(t).$$

Namely, we end up with

$$2\langle \lambda_{\hat{\theta}} - \lambda_\theta, \lambda_{\hat{\theta}} - \lambda\rangle_T = \langle \nabla R_T(\hat{\theta}), \hat{\theta} - \theta\rangle + \frac{2}{T}\sum_{j=1}^{d}\int_0^T (\lambda_{j,\hat{\theta}}(t) - \lambda_{j,\theta}(t))\,\mathrm{d}M_j(t). \quad \text{(II.23)}$$

The parallelogram identity gives

$$2\langle \lambda_{\hat{\theta}} - \lambda_\theta, \lambda_{\hat{\theta}} - \lambda\rangle_T = \|\lambda_{\hat{\theta}} - \lambda\|_T^2 + \|\lambda_{\hat{\theta}} - \lambda_\theta\|_T^2 - \|\lambda_\theta - \lambda\|_T^2,$$

where we put $\|f\|_T^2 = \langle f, f\rangle_T$. Let us point out that, in the case $\langle \lambda_{\hat\theta} - \lambda_\theta, \lambda_{\hat\theta} - \lambda\rangle_T < 0$, one obtains

$$\|\lambda_{\hat\theta} - \lambda\|_T^2 \le \|\lambda_\theta - \lambda\|_T^2,$$

which directly implies the inequality of the Theorem. Thus, from now on, let us assume that

$$\langle \lambda_{\hat\theta} - \lambda_\theta, \lambda_{\hat\theta} - \lambda\rangle_T \ge 0.$$

The first order condition for $\hat\theta \in \arg\min_\theta\{R_T(\theta) + \mathrm{pen}(\theta)\}$ gives

$$-\nabla R_T(\hat\theta) \in \partial\,\mathrm{pen}(\hat\theta).$$

Let $\hat\theta_\partial = -\nabla R_T(\hat\theta)$. Since the subdifferential is a monotone mapping, we have $\langle \hat\theta - \theta, \hat\theta_\partial - \theta_\partial\rangle \ge 0$ for any $\theta_\partial \in \partial\,\mathrm{pen}(\theta)$. Thus from (II.23), one gets $\forall \theta_\partial \in \partial\,\mathrm{pen}(\theta)$,

$$2\langle \lambda_{\hat\theta} - \lambda_\theta, \lambda_{\hat\theta} - \lambda\rangle_T \le -\langle \theta_\partial, \hat\theta - \theta\rangle + \frac{2}{T}\sum_{j=1}^{d}\int_0^T (\lambda_{j,\hat\theta}(t) - \lambda_{j,\theta}(t))\,\mathrm{d}M_j(t). \quad \text{(II.24)}$$

We need now to characterize the structure of the subdifferentials involved in $\mathrm{pen}(\theta)$, to describe θ_∂. If $g_1(\mu) = \sum_{j=1}^d \hat w_j|\mu_j|$, for $\hat w_j \ge 0$, we have

$$\partial g_1(\mu) = \Big\{\hat w \odot \mathrm{sign}(\mu) + \hat w \odot f : \|f\|_\infty \le 1, \mu \odot f = 0\Big\}. \quad \text{(II.25)}$$

If $g_2(\mathbb{A}) = \sum_{1\le j,j'\le d, 1\le k\le K} \hat{\mathbb{W}}_{j,j',k}|\mathbb{A}_{j,j',k}|$, for $\hat{\mathbb{W}}_{j,j',k} \ge 0$, we have

$$\partial g_2(\mathbb{A}) = \Big\{\hat{\mathbb{W}} \odot \mathrm{sign}(\mathbb{A}) + \hat{\mathbb{W}} \odot \mathbb{F} : \|\mathbb{F}\|_\infty \le 1, \mathbb{A} \odot \mathbb{F} = 0\Big\}. \quad \text{(II.26)}$$

Now let $\boldsymbol{A} = \mathrm{hstack}(\mathbb{A})$ and $\hat{\boldsymbol{A}} = \mathrm{hstack}(\hat{\mathbb{A}})$. Let us recall that if $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ is the SVD of $\boldsymbol{A}$, we have $\mathscr{P}_{\boldsymbol{A}}(\boldsymbol{B}) = \boldsymbol{P}_{\boldsymbol{U}}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{P}_{\boldsymbol{V}} - \boldsymbol{P}_{\boldsymbol{U}}\boldsymbol{B}\boldsymbol{P}_{\boldsymbol{V}}$ and $\mathscr{P}^\perp_{\boldsymbol{A}}(\boldsymbol{B}) = (\boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{U}})\boldsymbol{B}(\boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{V}})$ (projection onto the column and row space of $\boldsymbol{A}$ and projection onto its orthogonal space). Now, for $g_3(\boldsymbol{A}) = \hat\tau\|\boldsymbol{A}\|_*$, we have

$$\partial g_3(\boldsymbol{A}) = \Big\{\hat\tau\boldsymbol{U}\boldsymbol{V}^\top + \hat\tau\mathscr{P}^\perp_{\boldsymbol{A}}(\boldsymbol{F}) : \|\boldsymbol{F}\|_{\mathrm{op}} \le 1\Big\}, \quad \text{(II.27)}$$

see for instance [Lew95]. Now, write

$$-\langle \theta_\partial, \hat\theta - \theta\rangle = -\langle \mu_\partial, \hat\mu - \mu\rangle - \langle \mathbb{A}_{\partial,1}, \hat{\mathbb{A}} - \mathbb{A}\rangle - \langle \boldsymbol{A}_{\partial,*}, \hat{\boldsymbol{A}} - \boldsymbol{A}\rangle$$

with $\mu_\partial \in \partial g_1(\mu)$, $\mathbb{A}_{\partial,1} \in \partial g_2(\mathbb{A})$ and $\boldsymbol{A}_{\partial,*} \in \partial g_3(\boldsymbol{A})$. Using Equation (II.25), (II.26) and (II.27), we can write

$$\begin{aligned}-\langle \theta_\partial, \hat\theta - \theta\rangle = &-\langle \hat w \odot \mathrm{sign}(\mu), \hat\mu - \mu\rangle - \langle \hat w \odot f, \hat\mu - \mu\rangle \\ &- \langle \hat{\mathbb{W}} \odot \mathrm{sign}(\mathbb{A}), \hat{\mathbb{A}} - \mathbb{A}\rangle - \langle \hat{\mathbb{W}} \odot \mathbb{F}_1, \hat{\mathbb{A}} - \mathbb{A}\rangle \\ &- \hat\tau\langle \boldsymbol{U}\boldsymbol{V}^\top, \hat{\boldsymbol{A}} - \boldsymbol{A}\rangle - \hat\tau\langle \boldsymbol{F}_*, \mathscr{P}^\perp_{\boldsymbol{A}}(\hat{\boldsymbol{A}} - \boldsymbol{A})\rangle,\end{aligned}$$

where by duality between the norms $\|\cdot\|_1$ and $\|\cdot\|_\infty$, and between $\|\cdot\|_*$ and $\|\cdot\|_{\mathrm{op}}$, we can choose $f, \mathbb{F}_1$ and $\boldsymbol{F}_*$ such that

$$\langle \hat{w} \odot f, \hat{\mu} - \mu \rangle = \|(\hat{\mu} - \mu)_{\mathrm{supp}(\mu)^\perp}\|_{1,\hat{w}}, \quad \langle \hat{\mathbb{W}} \odot \mathbb{F}_1, \hat{\mathbb{A}} - \mathbb{A} \rangle = \|(\hat{\mathbb{A}} - \mathbb{A})_{\mathrm{supp}(\mathbb{A})^\perp}\|_{1,\hat{\mathbb{W}}}$$

and

$$\langle \boldsymbol{F}_*, \mathscr{P}_{\boldsymbol{A}}^\perp(\hat{\boldsymbol{A}} - \boldsymbol{A}) \rangle = \|\mathscr{P}_{\boldsymbol{A}}^\perp(\hat{\boldsymbol{A}} - \boldsymbol{A})\|_*,$$

which leads to

$$\begin{aligned} -\langle \theta_\partial, \hat{\theta} - \theta \rangle \leq{}& \|(\hat{\mu} - \mu)_{\mathrm{supp}(\mu)}\|_{1,\hat{w}} - \|(\hat{\mu} - \mu)_{\mathrm{supp}(\mu)^\perp}\|_{1,\hat{w}} \\ &+ \|(\hat{\mathbb{A}} - \mathbb{A})_{\mathrm{supp}(\mathbb{A})}\|_{1,\hat{\mathbb{W}}} - \|(\hat{\mathbb{A}} - \mathbb{A})_{\mathrm{supp}(\mathbb{A})^\perp}\|_{1,\hat{\mathbb{W}}} \\ &+ \hat{\tau}\|\mathscr{P}_{\boldsymbol{A}}(\hat{\boldsymbol{A}} - \boldsymbol{A})\|_* - \hat{\tau}\|\mathscr{P}_{\boldsymbol{A}}^\perp(\hat{\boldsymbol{A}} - \boldsymbol{A})\|_*. \end{aligned}$$

Now, we decompose the noise term of (II.24):

$$\begin{aligned} &\frac{2}{T}\sum_{j=1}^d \int_0^T (\lambda_{j,\hat{\theta}}(t) - \lambda_{j,\theta}(t))\,\mathrm{d}M_j(t) \\ &\quad = \frac{2}{T}\sum_{j=1}^d (\hat{\mu}_j - \mu_j)\int_0^T \mathrm{d}M_j(t) + \frac{2}{T}\sum_{\substack{1\leq j,j'\leq d \\ 1\leq k\leq K}} (\hat{a}_{j,j',k} - a_{j,j',k})\int_0^T \mathbb{H}_{j,j',k}(t)\,\mathrm{d}M_j(t) \\ &\quad = \frac{2}{T}\langle \hat{\mu} - \mu, M(T)\rangle + \frac{2}{T}\langle \hat{\mathbb{A}} - \mathbb{A}, \mathbb{Z}(T)\rangle, \end{aligned}$$

where $M(T) = [M_1(T) \cdots M_d(T)]^\top$ and where $\mathbb{Z}(T)$ is the $d \times d \times K$ tensor with entries

$$\mathbb{Z}_{j,j',k}(T) = \int_0^T \mathbb{H}_{j,j',k}(t)\,\mathrm{d}M_j(t).$$

Recall that hstack is the horizontally stacking operator defined by (II.15). The following upper bounds

$$\begin{aligned} &|\langle \hat{\mu} - \mu, M(T)\rangle| \leq \sum_{j=1}^d |\hat{\mu}_j - \mu_j||M_j(T)| \\ &|\langle \hat{\mathbb{A}} - \mathbb{A}, \mathbb{Z}(T)\rangle| \leq \sum_{\substack{1\leq j,j'\leq d \\ 1\leq k\leq K}} |\hat{\mathbb{A}}_{j,j',k} - \mathbb{A}_{j,j',k}||\mathbb{Z}_{j,j',k}(T)| \\ &|\langle \hat{\mathbb{A}} - \mathbb{A}, \mathbb{Z}(T)\rangle| = \langle \mathrm{hstack}(\hat{\mathbb{A}} - \mathbb{A}), \mathrm{hstack}(\mathbb{Z}(T))\rangle \leq \|\mathrm{hstack}(\mathbb{Z}(T))\|_{\mathrm{op}}\|\mathrm{hstack}(\hat{\mathbb{A}} - \mathbb{A})\|_*, \end{aligned}$$

entail that we need to upper bound the three terms

$$|M_j(T)|, \quad |\mathbb{Z}_{j,j',k}(T)| \quad \text{and} \quad \|\mathrm{hstack}(\mathbb{Z}(T))\|_{\mathrm{op}}$$

by data-driven quantities. Let us start with $\|\mathrm{hstack}(\mathbb{Z}(T))\|_{\mathrm{op}}$. Denote for short $\boldsymbol{Z}(t) = \mathrm{hstack}(\mathbb{Z}(t))$ and $\boldsymbol{H}(t) = \mathrm{hstack}(\mathbb{H}(t))$ where $\mathbb{H}(t)$ is defined by (II.14). We note that

$$\boldsymbol{Z}(t) = \int_0^t \mathrm{diag}(\mathrm{d}M(s))\boldsymbol{H}(s),$$

namely

$$(\boldsymbol{Z}(t))_{j,j'+(k-1)d} = \int_0^t (\mathbb{H}(t-s))_{j,j',k}\,\mathrm{d}M_j(s)$$

for any $1 \le j, j' \le d$ and $1 \le k \le K$. We need the following corollary.

Corollary 1. *The following deviation inequality holds*

$$\mathbb{P}\Big[\|\boldsymbol{Z}(t)\|_{\mathrm{op}} \ge 2\sqrt{\lambda_{\max}(\widehat{\boldsymbol{V}}(t))(x+\log(2d)+\ell(t))} + 14.39(x+\log(2d)+\ell(t))(1+\sup_{0\le s\le t}\|\boldsymbol{H}(s)\|_{\infty,2})\Big] \le 23.45e^{-x},$$

where

$$\lambda_{\max}(\widehat{\boldsymbol{V}}(t)) = \lambda_{\max}\Big(\int_0^t \boldsymbol{H}^\top(s)\boldsymbol{H}(s)\,\mathrm{diag}(\mathrm{d}N(s))\Big) \bigvee \max_{j=1,\dots,d}\int_0^t \|\boldsymbol{H}_{j,\bullet}(s)\|_2^2\,\mathrm{d}N_j(s),$$

and where

$$\ell(t) = 2\log\log\Big(\frac{4\lambda_{\max}(\widehat{\boldsymbol{V}}(t))}{x} \vee 2\Big) + 2\log\log\Big(4\sup_{0\le s\le t}\|\boldsymbol{H}(s)\|_{\infty,2} \vee 2\Big).$$

The proof of Corollary 1 is given in Section 8.6 below. Corollary 1 proves that $\frac{1}{T}\|\boldsymbol{Z}(t)\|_{\mathrm{op}} \le \frac{\hat{\tau}}{2}$ holds with probability $1-23.45e^{-x}$, with

$$\hat{\tau} = 4\sqrt{\frac{\lambda_{\max}(\widehat{\boldsymbol{V}}(T)/T)(x+\log(2d)+\ell(T))}{T}} + 28.78\frac{x+\log(2d)+\ell(T))(1+\sup_{0\le t\le T}\|\boldsymbol{H}(t)\|_{\infty,2})}{T},$$

which leads to the choice of $\hat{\tau}$ given in Section 4. This entails that, on an event of probability larger than $1-23.45e^{-x}$, we have

$$\frac{1}{T}|\langle\hat{\mathbb{A}}-\mathbb{A}, \mathbb{Z}(T)\rangle| \le \frac{\hat{\tau}}{2}\|\mathrm{hstack}(\hat{\mathbb{A}}-\mathbb{A})\|_*.$$

Using again Corollary 1 with $\boldsymbol{H}(t) \equiv 1$ (constant number equal to 1) and $M = M_j$ gives that $\frac{1}{T}|M_j(T)| \le \frac{\hat{w}_j}{3}$ for all $j = 1,\dots,d$ with probability $1-23.45e^{-x}$ with

$$\hat{w}_j = 6\sqrt{\frac{(N_j(T)/T)(x+\log d+\ell_j(T))}{T}} + 86.34\frac{x+\log d+\ell_j(T)}{T},$$

with $\ell_j(T) = 2\log\log(\frac{4N_j(T)}{x} \vee 2) + 2\log\log 4$. This entails that, on an event of probability larger than $1 - 23.45e^{-x}$, we have

$$\frac{2}{T}|\langle \hat{\mu} - \mu, M(T)\rangle| \leq \frac{2}{3}\|\hat{\mu} - \mu\|_{1,\hat{w}}.$$

Using a last time Corollary 1 with $\boldsymbol{H}(t) = H_{j,j',k}(t)$ and $M = M_j$ gives $\frac{1}{T}|\mathbb{Z}_{j,j',k}(T)| \leq \frac{\hat{\mathbb{W}}_{j,j',k}}{2}$ uniformly for j, j', k for

$$\hat{\mathbb{W}}_{j,j',k} = 4\sqrt{\frac{\frac{1}{T}\int_0^T H_{j,j',k}(t)^2\,\mathrm{d}N_j(t)(x + \log(Kd^2) + \mathbb{L}_{j,j',k}(T))}{T}} + 28.78\frac{(x + \log(Kd^2) + \mathbb{L}_{j,j',k}(T))(1 + \sup_{0\leq t\leq T}|H_{j,j',k}(t)|)}{T},$$

where

$$\mathbb{L}_{j,j',k}(T) = 2\log\log\Big(\frac{4\int_0^T H_{j,j',k}(t)^2\,\mathrm{d}N_j(t)}{x} \vee 2\Big) + 2\log\log\Big(4\sup_{0\leq t\leq T}|H_{j,j',k}(t)| \vee 2\Big),$$

which entails that on an event of probability larger than $1 - 23.45e^{-x}$, we have

$$\frac{1}{T}|\langle \hat{\mathbb{A}} - \mathbb{A}, \mathbb{Z}(T)\rangle| \leq \frac{1}{2}\|\hat{\mathbb{A}} - \mathbb{A}\|_{1,\hat{\mathbb{W}}}.$$

This entails that, with a probability larger than $1 - 3 \times 23.45e^{-x}$, one has

$$\begin{aligned}
0 \leq -\langle \theta_\partial, \hat{\theta} - \theta\rangle + \frac{2}{T}\sum_{j=1}^d \int_0^T (\lambda_{j,\hat{\theta}}(t) - \lambda_{j,\theta}(t))\,\mathrm{d}M_j(t) \\
&\leq \frac{5}{3}\|(\hat{\mu} - \mu)_{\mathrm{supp}(\mu)}\|_{1,\hat{w}} - \frac{1}{3}\|(\hat{\mu} - \mu)_{\mathrm{supp}(\mu)^\perp}\|_{1,\hat{w}} \\
&\quad + \frac{3}{2}\|(\hat{\mathbb{A}} - \mathbb{A})_{\mathrm{supp}(\mathbb{A})}\|_{1,\hat{\mathbb{W}}} - \frac{1}{2}\|(\hat{\mathbb{A}} - \mathbb{A})_{\mathrm{supp}(\mathbb{A})^\perp}\|_{1,\hat{\mathbb{W}}} \\
&\quad + \frac{3}{2}\hat{\tau}\|\mathscr{P}_{\boldsymbol{A}}(\hat{\boldsymbol{A}} - \boldsymbol{A})\|_* - \frac{1}{2}\hat{\tau}\|\mathscr{P}^\perp_{\boldsymbol{A}}(\hat{\boldsymbol{A}} - \boldsymbol{A})\|_*,
\end{aligned}$$

where we recall once again that $\boldsymbol{A} = \mathrm{hstack}(\mathbb{A})$ and $\hat{\boldsymbol{A}} = \mathrm{hstack}(\hat{\mathbb{A}})$. This matches the constraint of Definition 1 with $\mu' = \hat{\mu} - \mu$ and $\mathbb{A}' = \hat{\mathbb{A}} - \mathbb{A}$, so that it entails

$$\|(\hat{\mu} - \mu)_{\mathrm{supp}(\mu)}\|_2 \vee \|(\hat{\mathbb{A}} - \mathbb{A})_{\mathrm{supp}(\mathbb{A})}\|_F \vee \|\mathscr{P}_{\boldsymbol{A}}(\hat{\boldsymbol{A}} - \boldsymbol{A})\|_F \leq \kappa(\theta)\|\lambda_{\hat{\theta}} - \lambda_\theta\|_T. \tag{II.28}$$

Putting all this together gives

$$\begin{aligned}
&-\langle\theta_\partial,\hat\theta-\theta\rangle+\frac{2}{T}\langle\hat\mu-\mu,M(T)\rangle+\frac{2}{T}\langle\hat{\mathbb{A}}-\mathbb{A},\mathbb{Z}(T)\rangle\\
&\quad\le\frac{5}{3}\|(\hat\mu-\mu)_{\mathrm{supp}(\mu)}\|_{1,\hat w}-\frac{1}{3}\|(\hat\mu-\mu)_{\mathrm{supp}(\mu)^\perp}\|_{1,\hat w}\\
&\qquad+\frac{3}{2}\|(\hat{\mathbb{A}}-\mathbb{A})_{\mathrm{supp}(\mathbb{A})}\|_{1,\hat{\mathbb{W}}}-\frac{1}{2}\|(\hat{\mathbb{A}}-\mathbb{A})_{\mathrm{supp}(\mathbb{A})^\perp}\|_{1,\hat{\mathbb{W}}}\\
&\qquad+\frac{3}{2}\hat\tau\|\mathscr{P}_{\boldsymbol{A}}(\hat{\boldsymbol{A}}-\boldsymbol{A})\|_*-\frac{1}{2}\hat\tau\|\mathscr{P}^\perp_{\boldsymbol{A}}(\hat{\boldsymbol{A}}-\boldsymbol{A})\|_*\\
&\quad\le\frac{5}{3}\|(\hat w)_{\mathrm{supp}(\mu)}\|_2\|(\hat\mu-\mu)_{\mathrm{supp}(\mu)}\|_2+\frac{3}{2}\|(\hat{\mathbb{W}})_{\mathrm{supp}(\mathbb{A})}\|_F\|(\hat{\mathbb{A}}-\mathbb{A})_{\mathrm{supp}(\mathbb{A})}\|_F\\
&\qquad+\frac{3}{2}\hat\tau\sqrt{\mathrm{rank}(\boldsymbol{A})}\|\mathscr{P}_{\boldsymbol{A}}(\hat{\boldsymbol{A}}-\boldsymbol{A})\|_F,
\end{aligned}$$

where we used Cauchy-Schwarz's inequality. This finally gives

$$\begin{aligned}
\|\lambda_{\hat\theta}-\lambda\|_T^2\le\|\lambda_\theta-\lambda\|_T^2-\|\lambda_{\hat\theta}-\lambda_\theta\|_T^2\\
+\kappa(\theta)\Big(\frac{5}{3}\|(\hat w)_{\mathrm{supp}(\mu)}\|_2+\frac{3}{2}\|(\hat{\mathbb{W}})_{\mathrm{supp}(\mathbb{A})}\|_F+\frac{3}{2}\hat\tau\sqrt{\mathrm{rank}(\boldsymbol{A})}\Big)\|\lambda_{\hat\theta}-\lambda_\theta\|_T
\end{aligned}$$

where we used (II.28). The conclusion of the proof of Theorem 4 follows from the fact that $ax-x^2\le a^2/4$ for any $a,x>0$.

8.6 Proof of Corollary 1

We simply use Theorem 3. First, we remark that $\boldsymbol{Z}(t)=\int_0^t\mathbb{T}(s)\circ\mathrm{diag}(\mathrm{d}M(s))$ for the tensor $\mathbb{T}(t)$ of size $d\times Kd\times d\times d$ given by

$$(\mathbb{T}(t))_{i,j;k,l}=(\boldsymbol{I})_{i,k}(\boldsymbol{H}(t))_{l,j} \tag{II.29}$$

for $1\le i,k,l\le d$ and $1\le j\le Kd$. Note that we have

$$\mathbb{T}_{\bullet,\bullet;k,l}(t)=e_k\boldsymbol{H}_{l,\bullet}(t)^\top\quad\text{and}\quad\mathbb{T}_{\bullet,\bullet;k,l}(t)^\top=\boldsymbol{H}_{l,\bullet}(t)e_k^\top \tag{II.30}$$

where $e_k\in\mathbb{R}^d$ stands for the k-th element of the canonical basis of $\mathbb{R}^d$ and where $\boldsymbol{H}_{l,\bullet}(t)\in\mathbb{R}^{Kd}$ stands for the vector corresponding to the l-th row of the matrix $\boldsymbol{H}(t)$. Therefore, we have

$$\mathbb{T}_{\bullet,\bullet;k,l}(t)\mathbb{T}^\top_{\bullet,\bullet;k,l}(t)=\|\boldsymbol{H}_{l,\bullet}(t)\|_2^2e_ke_k^\top\quad\text{and}\quad\mathbb{T}^\top_{\bullet,\bullet;k,l}(t)\mathbb{T}_{\bullet,\bullet;k,l}(t)=\boldsymbol{H}_{l,\bullet}(t)\boldsymbol{H}_{l,\bullet}(t)^\top$$

and therefore

$$\|\mathbb{T}_{\bullet,\bullet;k,l}(t)\|_{\mathrm{op}}=\sqrt{\lambda_{\max}(\mathbb{T}_{\bullet,\bullet;k,l}(t)\mathbb{T}^\top_{\bullet,\bullet;k,l}(t))}=\|\boldsymbol{H}_{l,\bullet}(t)\|_2$$

and

$$\|\mathbb{T}(t)\|_{\mathrm{op};\infty} = \max_{1\le l\le d} \|\boldsymbol{H}_{l,\bullet}(t)\|_2 = \|\boldsymbol{H}(t)\|_{\infty,2}.$$

One can prove in the same way that $\|\mathbb{T}^\top(t)\|_{\mathrm{op};\infty} = \|\boldsymbol{H}(t)\|_{\infty,2}$, so that for this choice of tensor $\mathbb{T}(t)$, we have $b_{\mathbb{T}}(t) = \|\boldsymbol{H}(t)\|_{\infty,2}$. Now, let us explicit what $\widehat{\boldsymbol{V}}_{\mathbb{T}}(t)$ is for the tensor (II.29). First, let us remind that

$$\widehat{\boldsymbol{V}}_{\mathbb{T}}(t) = \begin{bmatrix} \int_0^t \mathbb{T}(s)\mathbb{T}^\top(s) \circ \mathrm{diag}(\mathrm{d}N(s)) & \boldsymbol{0} \\ \boldsymbol{0} & \int_0^t \mathbb{T}^\top(s)\mathbb{T}(s) \circ \mathrm{diag}(\mathrm{d}N(s)) \end{bmatrix}.$$

Using (II.30) we get

$$(\mathbb{T}(t)\mathbb{T}(t)^\top)_{\bullet,\bullet;,k,l} = e_k \boldsymbol{H}_{l,\bullet}(t)^\top \boldsymbol{H}_{l,\bullet}(t) e_k^\top = \|\boldsymbol{H}_{l,\bullet}(t)\|_2^2 e_k e_k^\top$$

so that $\int_0^t (\mathbb{T}(s)\mathbb{T}^\top(s)) \circ \mathrm{diag}(\mathrm{d}N(s))$ is the diagonal matrix with entries

$$\Big(\int_0^t (\mathbb{T}(s)\mathbb{T}^\top(s)) \circ \mathrm{diag}(\mathrm{d}N(s))\Big)_{j,j} = \int_0^t \|\boldsymbol{H}_{j,\bullet}(s)\|_2^2 \,\mathrm{d}N_j(s),$$

or equivalently

$$\int_0^t (\mathbb{T}(s)\mathbb{T}^\top(s)) \circ \mathrm{diag}(\mathrm{d}N(s)) = \int_0^t \mathrm{diag}(\boldsymbol{H}^\top(s)\boldsymbol{H}(s)) \,\mathrm{diag}(\mathrm{d}N(s)).$$

Using again (II.30) we get

$$(\mathbb{T}^\top(t)\mathbb{T}(t))_{\bullet,\bullet;,k,l} = \boldsymbol{H}_{l,\bullet}(t) e_k^\top e_k \boldsymbol{H}_{l,\bullet}(t)^\top = \boldsymbol{H}_{l,\bullet}(t)\boldsymbol{H}_{l,\bullet}(t)^\top$$

so that $\int_0^t (\mathbb{T}^\top(s)\mathbb{T}(s)) \circ \mathrm{diag}(\mathrm{d}N(s))$ is the matrix with entries

$$\Big(\int_0^t (\mathbb{T}^\top(s)\mathbb{T}(s)) \circ \mathrm{diag}(\mathrm{d}N(s))\Big)_{i,j} = \sum_{l=1}^d \int_0^t \boldsymbol{H}_{l,i}(s)\boldsymbol{H}_{l,j}(s) \,\mathrm{d}N_l(s)$$

or equivalently

$$\int_0^t (\mathbb{T}^\top(s)\mathbb{T}(s)) \circ \mathrm{diag}(\mathrm{d}N(s)) = \int_0^t \boldsymbol{H}^\top(s)\boldsymbol{H}(s) \,\mathrm{diag}(\mathrm{d}N(s)).$$

Finally, we obtain that

$$\lambda_{\max}(\widehat{\boldsymbol{V}}_t) = \lambda_{\max}\Big(\int_0^t \boldsymbol{H}^\top(s)\boldsymbol{H}(s) \,\mathrm{diag}(\mathrm{d}N(s))\Big) \bigvee \max_{j=1,\dots,d} \int_0^t \|\boldsymbol{H}_{j,\bullet}(t)\|_2^2 \,\mathrm{d}N_j(s).$$

This concludes the proof of the corollary. □

CHAPTER III

Background on first order composite sum minimization

1 Composite sum minimization

A wide variety of machine learning tasks consist in optimizing the following objective

$$\min_{w\in\mathbb{R}^d} F(w) \quad \text{with} \quad F(w) = f(w) + g(w), \quad f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w), \tag{III.1}$$

where each convex function f_i typically corresponds to a loss associated to the i-th observation of a dataset of n samples and the convex function g is a penalization term. This framework includes classification with logistic regression with $f_i(w) = \log(1+\exp(-y_i w^\top x_i))$, least square regression with $f_i(w) = (y_i - w^\top x_i)^2$ among many others. It is common to assume that function f is gradient-Lipschitz, namely $\|\nabla f(x) - \nabla f(y)\| \leq L\|x-y\|$ for any $x, y \in \mathbb{R}^d$ where $\|.\|$ stands for the Euclidean norm on $\mathbb{R}^d$, and $L > 0$ is the Lipschitz constant. Besides, we sometimes consider that the penalization terms g is strongly convex, that is $\|\nabla g(x) - \nabla g(y)\| \geq \mu\|x-y\|$ for any $x, y \in \mathbb{R}^d$ with $\mu > 0$. In this case, the condition number $\kappa = \frac{L}{\mu}$ is a parameter of interest used to express the average number of operations needed to reach an ε-precise solution w such that $F(w) \leq F(w^*) + \varepsilon$ where $\varepsilon > 0$ and w^* is the minimizer of (III.1). In this chapter we give an overview of the existing first order algorithms used for composite sum minimization. By definition, these algorithms require evaluation of gradients, a technique that dates back to 1847 [Cau47]. We first present batch methods that compute the gradient of f at each iteration. Then we discuss about stochastic methods that compute the gradient of only one f_i at each iteration and hence performs faster but less efficient updates. We finish with variance reduced stochastic algorithms that combine the best of both worlds by performing fast and efficient updates.

2 Batch gradient descent

Batch gradient descent methods compute at each step $t \geq 1$ the gradient $\nabla f(w^t)$ to determine the next iterate $w^{(t+1)}$. When f is gradient-Lipschitz, this choice is justified by the following *descent lemma* [Ber99, Proposition A.24] ensuring that

$$f(w^{t+1}) \leq f(w^t) + (w^{t+1} - w^t)^\top \nabla f(w^t) + \frac{L}{2}\|w^{t+1} - w^t\|^2$$

for any $w^{(t+1)}, w^t \in \mathbb{R}^d$. In light of this lemma the optimal value w^{t+1} leading the bigger gain while minimizing $f(w^{t+1})$ is $w^t - \frac{1}{L}\nabla f(w^t)$. However, the full objective also includes a penalization term g which leads to rather find the minimizer of the following quadratic approximation [BT09]

$$w^{(t+1)} = \operatorname*{argmin}_{w \in \mathbb{R}^d} f(w^t) + (w - w^t)^\top \nabla f(w^t) + \frac{L}{2}\|w - w^t\|^2 + g(w),$$

which writes

$$w^{(t+1)} = \operatorname{prox}_{\frac{1}{L}g}\left(w^t - \frac{1}{L}\nabla f(w^t)\right),$$

where prox_g stands for the proximal operator [CP11] associated to g defined as following.

Definition 1. Proximal operator. *For a convex function $g : \mathscr{D}_g \to \mathbb{R}$, the proximal operator associated to g is given by*

$$\operatorname{prox}_g(y) = \operatorname*{argmin}_{x \in \mathscr{D}_g}\left(\frac{1}{2}\|y - x\|^2 + g(x)\right).$$

The proximal operator always exists and is uniquely defined as the minimizer of a strictly convex function.

Performing successively this update leads to ISTA algorithm [BT09] and is described in Algorithm III.1. When the Lipschitz constant L is known, the step size parameter

Algorithm III.1 ISTA

Require: Starting point w^0, step size $\eta > 0$
  for $t = 0, 1, \ldots$ **do**
    $w^{(t+1)} \leftarrow \operatorname{prox}_{\eta g}\left(w^t - \eta \nabla f(w^t)\right)$
  end for

η is set to the optimal value $\frac{1}{L}$ that leads to the biggest guaranteed gain. When it is unknown, it is set using, at each step, backtracking line search which amounts to find the bigger step size η whose inverse would be a suitable Lipschitz constant

Table III.1: Convergence rates of first order batch methods with optimal step size. κ is the condition number.

	Gradient-Lipschitz	Gradient-Lipschitz and strong convexity
ISTA	$\mathcal{O}(L/t)$	$\mathcal{O}\left(\left(1-\kappa^{-1}\right)^t\right)$
FISTA	$\mathcal{O}(L/t^2)$	$\mathcal{O}\left(\left(1-\kappa^{-1/2}\right)^t\right)$

candidate in the sense that it verifies the descent lemma. ISTA also has an accelerated version called FISTA [BT09]. FISTA is very similar but instead of performing gradient descent from the last reached point it rather performs it from a momentum, that is a linear combination of the last two reached points. It is described in Algorithm III.2. We compare the efficiency of these algorithms with their convergence rates $\rho(t)$, that

Algorithm III.2 FISTA

Require: Starting point w^0, step size $\eta > 0$
  $z^1 \leftarrow w^0$; $\alpha^1 = 1$
  for $t = 1, 2, \ldots$ **do**
    $w^t \leftarrow \text{prox}_{\eta g}(z^t - \eta \nabla f(z^t))$
    $\alpha^{(t+1)} \leftarrow \frac{1+\sqrt{1+4\alpha^{t^2}}}{2}$
    $z^{(t+1)} \leftarrow w^t + \frac{\alpha^t - 1}{\alpha^{(t+1)}}(w^t - w^{(t-1)})$
  end for

is the value that bounds the difference between the current objective and the optimal objective after t passes on the dataset. Formally, $\rho(t)$ is such that $F(w^t) \leq F(w^*) + \rho(t)$. The convergence rate is intrinsically linked to number of iterations needed to reach an ε-precise solution. FISTA and ISTA convergence rates are reported in Table III.1. In both cases, $\rho(t)$ can be expressed as ρ^t where $0 < \rho < 1$. These rate of convergence are called linear. Also, the rate of convergence of FISTA is known to be optimal for batch first order algorithms [Nes83].

3 Stochastic gradient descent

In a big data setting, a full gradient $\nabla f(w^t)$ is computationally expensive. However, with its structure, this problem can also be considered as an accumulation of smaller problems f_i for $i = 1, \ldots n$ that have a common behavior. Stochastic gradient descent (SGD) [RM51] exploits this and instead of computing the full gradient $\nabla f(w^t)$ at each step, uses a random variable $\phi^t \in \mathbb{R}^d$ such that $\mathbb{E}[\phi^t] = \nabla f(w^t)$. Generally, at each iteration we sample a random index i in $\{1, \ldots, n\}$ and compute the gradient

of the associated loss $\nabla f_i(w^t)$ which is an unbiased estimator of the full gradient. This procedure is detailed in Algorithm III.3. The iterate w^t is updated n times at

Algorithm III.3 Stochastic gradient descent

Require: Starting point w^0, a sequence of step size $(\eta_t)_{t\geq 0}$
  for $t = 0, 1, \ldots$ **do**
    Sample at random i in $\{1, \ldots, n\}$
    $w^{(t+1)} \leftarrow \text{prox}_{\eta_t g}\left(w^t - \eta_t \nabla f_i(w^t)\right)$
  end for

each pass on the dataset while it is updated only once with batch methods. This leads to much faster iterations. However, this method does not converge easily to a precise solution because $\nabla f_i(w^t)$ does not approach zero when w^t is close to the optimal value w^*. Hence, it keeps oscillating unless the sequence of step size $(\eta_t)_{t\geq 0}$ is decreasing, which eventually affects the convergence speed. In practice, the choice of this sequence is critical but also difficult. Driven by deep learning where SGD is widely used [RHW86], many variants have been introduced to address this issue such as Momentum [Qia99], AdaGrad [DHS11], ADADELTA [Zei12] or Adam [KB15]. These methods are more adaptive and less sensitive to the step size choice. Theoretically, for gradient-Lipschitz and strongly convex losses, SGD reaches a convergence rate in $\mathcal{O}\left(\frac{\kappa}{t}\right)$ where each iteration is approximatively n times faster than in the batch case. Thus, SGD typically converges very quickly during the first iterations but then slows down and has difficulties finding a very precise solution. Indeed its convergence rate is not linear, thus the number of iterations to reach an ε-precision solution is in $\mathcal{O}(1/\varepsilon)$ instead of $\mathcal{O}(\log 1/\varepsilon)$.

4 Variance reduced stochastic gradient descent

Recently, new stochastic solvers apply Monte-Carlo techniques of variance reduction to stochastic gradient descent. [SLRB17], [SSZ13], [JZ13], [DBLJ14]. As illustrated in Figure III.1, these algorithms combine the best of both worlds: quick iterations (each update has a complexity comparable to SGD) and the precision of batch methods. They rely on variance control which we will first formalize as a more general variance reduction technique as in [DBLJ14].

Let X be a random variable and Y another random variable hopefully correlated with X. To obtain a (possibly biased) estimation of $\mathbb{E}[X]$, we can set the following random variable Z for any $\alpha \in [0, 1]$ to

$$Z = \alpha(X - Y) + \mathbb{E}[Y]. \tag{III.2}$$

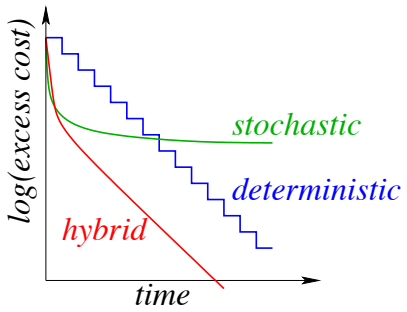

Figure III.1: Illustration of the convergence of variance reduced stochastic algorithms (hybrid) that combine the fast iterations of SGD (stochastic) and the precision of batch (deterministic) methods. From Francis Bach lectures.

If $\alpha = 1$, $\mathbb{E}[Z]$ is an unbiased estimation of $\mathbb{E}[X]$. Its variance equals

$$\mathrm{Var}[Z] = \alpha^2\big(\mathrm{Var}[X] + \mathrm{Var}[Y]) - 2\mathrm{Cov}(X, Y)\big).$$

$\mathrm{Var}[Z]$ is smaller than $\mathrm{Var}[X]$ if X and Y are sufficiently correlated. Variance reduced algorithms uses this technique to lower the variance of ϕ^t, the estimator of the full gradient $\nabla f(w^t)$. With this technique, ϕ^t can tend to 0 when w^t approaches w^* and the algorithm can then use constant step sizes in contrary to SGD. Three variance reduction algorithms fall directly into this framework.

SAG and SAGA The SAG algorithm [SLRB17] was the first method to obtain in 2012 a linear convergence rate with only one gradient ∇f_i computed per iteration. It works by storing the last computed gradient of each observation $\nabla f_i(w^{t_i})$ where t_i is the last iteration at which the observation i was seen. Formally, like in SGD, the random variable X of (III.2) is still $\nabla f_i(w^t)$ but now Y is set to $\nabla f_i(w^{t_i})$ whose expectation is $\frac{1}{n}\sum_{i=1}^n f_i(w^{t_i})$. However, SAG used a biased estimation of the full gradient ($\alpha = \frac{1}{n}$) and could not handle the composite case. SAGA [DBLJ14] is the unbiased version of SAG ($\alpha = 1$) and can theoretically handle the composite case with possibly non smooth penalizations g. Its implementation is detailed in Algorithm III.4. SAG [SLRB17] algorithm is identical to SAGA but the descent direction in the 7-th line is modified to $\nabla f_i(w^t) - \psi_i + \Psi$ instead of $\nabla f_i(w^t) - \psi_i + \frac{1}{n}\Psi$. Both algorithms obtain a linear rate in the strongly convex case (reported in Table III.2) with only one gradient ∇f_i computed at each iteration. This is obtained with the extra cost of maintaining n gradients in memory. This memory cost is generally in $\mathcal{O}(nd)$ where d is the size of the vector w but is much lower for the widely used generalized linear models. Indeed, in this family of models the loss writes $f_i(w^t) = \varphi_i(w^{t\top} x_i)$ where $\varphi_i : \mathbb{R} \to \mathbb{R}$ and $x_i \in \mathbb{R}^d$ is the i-th observation of the dataset. Hence, their gradients

Algorithm III.4 SAGA

Require: Starting point w^0, a step size $\eta > 0$, ψ_i for $i = 1, \dots, n$
  $\Psi \leftarrow \sum_{i=1}^{n} \psi_i$
  for $t = 0, 1, \dots$ **do**
    Sample at random i in $\{1, \dots, n\}$
    $w^{t+1} \leftarrow \operatorname{prox}_{\eta g}\left(w^t - \eta\left(\nabla f_i(w^t) - \psi_i + \frac{1}{n}\Psi\right)\right)$
    $\Psi \leftarrow \Psi - \psi_i$
    $\psi_i \leftarrow \nabla f_i(w^t)$
    $\Psi \leftarrow \Psi + \psi_i$
  end for

writes $\nabla f_i(w^t) = \varphi_i'(w^{t\top} x_i) x_i$, so storing the scalar $\varphi_i'(w^{t\top} x_i)$ is sufficient to retrieve the gradient $\nabla f_i(w^t)$ at a later time. The storage cost for this family (including linear regression and logistic regression among others) is then lowered from $\mathcal{O}(nd)$ to $\mathcal{O}(n)$. Finally, it is worth mentioning that the two algorithms are adaptable to the non strongly convex case, meaning that, with the same step size η, they obtain convergence guarantees in both cases.

SVRG Instead of storing n gradients, SVRG [JZ13] reduces variance by computing and storing, every m iterations, the full gradient of the current iterate $\nabla f(\widetilde{w})$. Associated to this stored gradient, Y is set to $\nabla f_i(\widetilde{w})$ (whose expectation is $\nabla f(\widetilde{w})$) X to $\nabla f_i(w^t)$ and α to 1. The proximal variant of SVRG [XZ14] is detailed in Algorithm III.5 and its convergence rate is reported in Table III.2. The proof of SVRG

Algorithm III.5 SVRG

Require: Starting point w^0, a step size $\eta > 0$
  for $t = 0, 1, \dots$ **do**
    if t is a multiple of m **then**
      $\widetilde{w} \leftarrow w^t$
      $\widetilde{\mu} \leftarrow \nabla f(\widetilde{w})$
    end if
    Sample at random i in $\{1, \dots, n\}$
    $w^{t+1} \leftarrow \operatorname{prox}_{\eta g}\left(w^t - \eta\left(\nabla f_i(w^t) - \nabla f_i(\widetilde{w}) + \widetilde{\mu}\right)\right)$
  end for

is much simpler and shorter than SAG and SAGA ones and manages to reach linear convergence rate without any extra storage cost but a gradient of size d. However, it also necessitates more computations as each iteration requires on average $2 + n/m$

gradients computation which is more than the single computation made by SAG and SAGA.

SDCA The last algorithm, SDCA [SSZ13] does not fall directly into this framework. It consists in finding the dual matrix $A \in \mathbb{R}^{n\times d}$ that maximizes the following dual objective, namely

$$\max_{A\in\mathbb{R}^{n\times d}} \frac{1}{n}\sum_{i=1}^{n} -f_i^*\big(-(A)_i\big) - g^*\left(\frac{1}{\lambda n}\sum_{i=1}^{n}(A)_i\right),$$

where $(A)_i \in \mathbb{R}^d$ is the i-th line of the matrix for $i = 1,\dots,n$. The optimal primal vector w^* is linked to the optimal dual matrix A^* through the the Karush–Kuhn–Tucker conditions with the following relation

$$w^* = \nabla g^*\left(\frac{1}{n}\sum_{i=1}^{n}(A^*)_i\right),$$

used to convert a dual iterate into a corresponding primal iterate as described in Algorithm III.6. Generally SDCA is presented in a generalized linear model context

Algorithm III.6 SDCA

Require: Starting dual A^0
  $v^0 \leftarrow \frac{1}{n}\sum_{i=1}^{n}(A^0)_i$
  for $t = 0, 1, \dots$ **do**
    Sample at random i in $\{1,\dots,n\}$
    Find $u \in \mathbb{R}^d$ that maximizes $-\frac{1}{n}f_i^*\big(-u\big) - g^*\big(v^t + \frac{1}{n}\big(u - (A^t)_i\big)\big)$
    $v^{t+1} \leftarrow v^t + \frac{1}{n}\big(u - (A^t)_i\big)$
    $(A^{t+1})_i \leftarrow u$
  end for
  $w^t \leftarrow \nabla g^*(v^t)$

and similarly as SAG and SAGA it lowers the number of new stored parameters from $\mathcal{O}(nd)$ to $\mathcal{O}(d)$. Indeed, in this context, each line $(A)_i$ of the matrix A writes $a_i x_i$ where $a_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^d$ is the i-th observation in the dataset. In this case, the dual objective is optimized over a dual vector $a \in \mathbb{R}^n$ and the optimization step of the 4-th line is a one dimensional problem. Also, when the penalization has the following decomposition $g(w) = h(w) + \frac{1}{2}\|w\|^2$ where prox_h exists, this optimization step can be much simplified along with the relation linking the primal and the dual iterates [SSZ14]. In addition, SDCA has no step size η which makes it easier to tune and benefits of its primal-dual approach to compute the duality gap which is an upper bound of the precision level ε [BV04]. Finally, while SDCA needs strong convexity to

Table III.2: Convergence rates of stochastic first order methods with optimal step size. Note that each iteration is $\mathcal{O}(n)$ times faster than batch algorithms of Table III.1. Also, as theses algorithms are stochastic, these rates are given on expectation. Only SGD, SAG and SAGA provides theoretical rate for non strongly convex problems. SVRG rate is computed for a learning rate $\eta = 0.1/L$ and is valid when t is a multiple of m.

	Gradient-Lipschitz	Gradient-Lipschitz and strong convexity
SGD	$\mathcal{O}\left(L/\sqrt{t}\right)$	$\mathcal{O}\left(\kappa/t\right)$
SAG	$\mathcal{O}\left(n/t\right)$	$\mathcal{O}\left(\left(1-\min(\frac{1}{16}\kappa^{-1}, \frac{1}{8}n^{-1})\right)^t\right)$
SAGA	$\mathcal{O}\left(n/t\right)$	$\mathcal{O}\left(\left(1-\min(\frac{1}{3}\kappa^{-1}, \frac{1}{4}n^{-1})\right)^t\right)$
SVRG	—	$\mathcal{O}\left(\left(12.5\kappa/m+0.25\right)^{t/m}\right)$
SDCA	—	$\mathcal{O}\left((1-(n+\kappa)^{-1})^t\right)$

derive a convergence rate, it also provides convergence guarantees on functions that are Lipschitz but not gradient-Lipschitz. Even if SDCA seems quite different from SAG, SAGA and SVRG it follows the same logic to reduce the variance of the iterates when we transpose its update in the primal [JZ13, DBLJ14, SS16].

5 Numerical comparison

The `tick` library (see Chapter V) is very handy to run experiment for convex optimization algorithms, as it allows to combine different bricks of models (such as linear or least squares regression), proximal operators and convex optimizers. In the following example, the model (logistic regression) and the proximal operator (elastic net) are fixed and define a convex objective $F(w) = f(w) + g(w)$ to minimize with simulated data. To perform this task, we compare the efficiency of six different optimization algorithms introduced before namely ISTA, FISTA [BT09], SGD [RM51] SVRG [JZ13], SAGA [DBLJ14] and SDCA [SSZ13] for which the free parameters were set to their theoretical optimal values.

```
from tick.linear_model import ModelLogReg, SimuLogReg
from tick.simulation import weights_sparse_gauss
from tick.solver import GD, AGD, SGD, SVRG, SAGA, SDCA
from tick.prox import ProxL2Sq
from tick.plot import plot_history

n_samples, n_features, = 5000, 50
weights0 = weights_sparse_gauss(n_features, nnz=10)
X, y = SimuLogReg(weights=weights0, n_samples=n_samples).simulate()

model = ModelLogReg().fit(X, y)
prox = ProxL2Sq(strength=1e-2)

w0 = np.zeros(model.n_coeffs)

ista = GD(linesearch=False).set_model(model).set_prox(prox)
ista.solve(w0, step=1. / model.get_lip_best())

fista = AGD(linesearch=False).set_model(model).set_prox(prox)
fista.solve(w0, step=1. / model.get_lip_best())

sgd = SGD().set_model(model).set_prox(prox)
sgd.solve(w0, step=60.)

svrg = SVRG().set_model(model).set_prox(prox)
svrg.solve(w0, step=1. / model.get_lip_max())

saga = SAGA().set_model(model).set_prox(prox)
saga.solve(x0, step=1 / model.get_lip_max())

sdca = SDCA(prox.strength).set_model(model)
sdca.solve()

plot_history([ista, fista, sgd, svrg, saga, sdca],
             log_scale=True, dist_min=True)
```

In Figure III.2, we show the evolution of the distance to the minimum, that is $F(w^t) - F(w^*)$, regarding the number of data passes. Note that the convergence rate $\rho(t)$ is an upper bound of the optimality gap. First, we observe that SGD makes a quick start but is then slowed down due to its poor convergence rate. Then, let's recall that a linear convergence rate ($\rho(t) = \rho^t$) is expected to be rendered by a straight line of slope $\log \rho$ on this graph because the y-axis is in log-scale. Hence, we observe these straight lines on ISTA, FISTA, SVRG SAGA and SDCA optimality gaps. Note that FISTA shows characteristics leaps that reminds us that it is not a descent method, meaning its objective is not guaranteed to decrease at each iteration. Finally, SVRG,

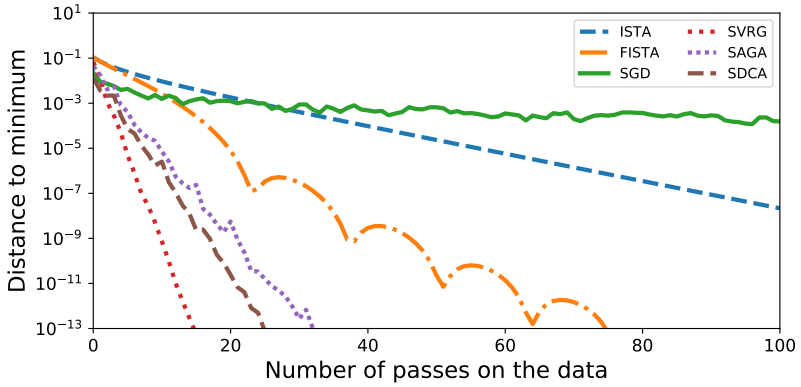

Figure III.2: Comparison of six optimization algorithms for logistic regression given the optimality gap. SVRG, SAGA and SDCA have the best properties as they combine the quick start of SGD with the best linear convergence rates.

SAGA and SDCA have the best properties as they combine the quick start of SGD with the best linear convergence rates.

CHAPTER IV

Dual optimization for convex constrained objectives without the gradient-Lipschitz assumption

Abstract

The minimization of convex objectives coming from linear supervised learning problems, such as penalized generalized linear models, can be formulated as finite sums of convex functions. For such problems, a large set of stochastic first-order solvers based on the idea of variance reduction are available and combine both computational efficiency and sound theoretical guarantees (linear convergence rates) [JZ13], [SLRB17], [SSZ13], [DBLJ14]. Such rates are obtained under both gradient-Lipschitz and strong convexity assumptions. Motivated by learning problems that do not meet the gradient-Lipschitz assumption, such as linear Poisson regression, we work under another smoothness assumption, and obtain a linear convergence rate for a shifted version of Stochastic Dual Coordinate Ascent (SDCA) [SSZ13] that improves the current state-of-the-art. Our motivation for considering a solver working on the Fenchel-dual problem comes from the fact that such objectives include many linear constraints, that are easier to deal with in the dual. Our approach and theoretical findings are validated on several datasets, for Poisson regression and another objective coming from the negative log-likelihood of the Hawkes process, which is a family of models which proves extremely useful for the modeling of information propagation in social networks and causality inference [DVG+16], [FWR+15].

1 Introduction

In the recent years, much effort has been made to minimize strongly convex finite sums with first order information. Recent developments, combining both numerical efficiency and sound theoretical guarantees, such as linear convergence rates, include SVRG [JZ13], SAG [SLRB17], SDCA [SSZ13] or SAGA [DBLJ14] to solve the following problem:

$$\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \varphi_i(w) + \lambda g(w), \tag{IV.1}$$

where the functions φ_i correspond to a loss computed at a sample i of the dataset, and g is a (eventually non-smooth) penalization. However, theoretical guarantees about these algorithms, such as linear rates guaranteeing a numerical complexity $\mathcal{O}(\log(1/\varepsilon))$ to obtain a solution ε-distant to the minimum, require both strong convexity of $\frac{1}{n}\sum_{i=1}^{n}\varphi_i + \lambda g$ and a gradient-Lipschitz property on each φ_i, namely $\|\varphi_i'(x) - \varphi_i'(y)\| \leq L_i\|x-y\|$ for any $x, y \in \mathbb{R}^d$, where $\|\cdot\|$ stands for the Euclidean norm on $\mathbb{R}^d$ and $L_i > 0$ is the Lipschitz constant. However, some problems, such as the linear Poisson regression, which is of practical importance in statistical image reconstruction among others (see [BBDV09] for more than a hundred references) do not meet such a smoothness assumption. Indeed, we have in this example $\varphi_i(w) = w^\top x_i - y_i \log(w^\top x_i)$ for $i = 1,\dots,n$ where $x_i \in \mathbb{R}^d$ are the features vectors and $y_i \in \mathbb{N}$ are the labels, and where the model-weights must satisfy the linear constraints $w^\top x_i > 0$ for all $i = 1,\dots,n$.

Motivated by machine learning problems described in Section 4 below, that do not satisfy the gradient-Lipschitz assumption, we consider a more specific task relying on a new smoothness assumption. Given convex functions $f_i : \mathscr{D}_f \to \mathbb{R}$ with $\mathscr{D}_f = (0, +\infty)$ such that $\lim_{t\to 0} f_i(t) = +\infty$, a vector $\psi \in \mathbb{R}^d$, features vectors $x_1,\dots,x_n \in \mathbb{R}^d$ corresponding to the rows of a matrix X we consider the objective

$$\min_{w\in\Pi(X)} P(w) \quad \text{where} \quad P(w) = \psi^\top w + \frac{1}{n}\sum_{i=1}^{n} f_i(w^\top x_i) + \lambda g(w), \tag{IV.2}$$

where $\lambda > 0$, $g : \mathbb{R}^d \to \mathbb{R}$ is a 1-strongly convex function and $\Pi(X)$ is the open polytope

$$\Pi(X) = \{w \in \mathbb{R}^d \,:\, \forall i \in \{1,\dots,n\},\ w^\top x_i > 0\}, \tag{IV.3}$$

that we assume to be non-empty. Note that the linear term $\psi^\top w$ can be included in the regularization g but the problem stands clearer if it is kept out.

Definition 1. *We say that a function $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ is* L-log smooth*, where $L > 0$, if it is a differentiable and strictly monotone convex function that satisfies*

$$|f'(x) - f'(y)| \leq \frac{1}{L} f'(x) f'(y) |x - y|$$

for all $x, y \in \mathscr{D}_f$.

We detail this property and its specificities in Section 2. All along the chapter, we assume that the functions f_i are L_i-log smooth. Note also that the Poisson regression objective fits in this setting, where $f_i(x) = -y_i \log x$ is y_i-log smooth and $\psi = \frac{1}{n}\sum_{i=1}^n x_i$. See Section 4.1 below for more details.

Related works. Standard first-order batch solvers (non stochastic) for composite convex objectives are ISTA and its accelerated version FISTA [BT09] and first-order stochastic solvers are mostly built on the idea of Stochastic Gradient Descent (SGD) [RM51]. Recently, stochastic solvers based on a combination of SGD and the Monte-Carlo technique of variance reduction [SLRB17], [SSZ13], [JZ13], [DBLJ14] turn out to be both very efficient numerically (each update has a complexity comparable to vanilla SGD) and very sound theoretically, because of strong linear convergence guarantees, that match or even improve the one of batch solvers. These algorithms involve gradient steps on the smooth part of the objective and theoretical guarantees justify such steps under the gradient-Lipschitz assumptions thanks to the descent lemma [Ber99, Proposition A.24]. Without this assumption, such theoretical guarantees fall apart. Also, stochastic algorithms loose their numerical efficiency if their iterates are projected on the feasible set $\Pi(X)$ at each iteration as Equation (IV.2) requires. STORC [HL16] can deal with constrained objectives without a full projection but is restricted to compact sets of constraints which is not the case of $\Pi(X)$. Then, a modified proximal gradient method from [TDKC15] provides convergence bounds relying on self-concordance [Nes13] rather than the gradient-Lipschitz property. However, the convergence rate is guaranteed only once the iterates are close to the optimal solution and we observed in practice that this algorithm is simply not working (since it ends up using very small step-sizes) on the problems considered here. Recently, [LFN18] has provided new descent lemmas based on *relative-smoothness* that hold on a wider set of functions including Poisson regression losses. This work is an extension of [BB16] that presented the same algorithm and detailed its application to Poisson regression losses. While this is more generic than our work, they only manage to reach sublinear convergence rates $\mathcal{O}(1/t)$ that applies only on positive solution (namely $w^* \in [0,\infty)^d$) while we reach linear rates for any solution $w^* \in \mathbb{R}^d$.

Our contribution. The first difficulty with the objective (IV.2) is to remain in the open polytope $\Pi(X)$. To deal with simpler constraints we rather perform optimization on the dual problem

$$\max_{\alpha \in -\mathscr{D}_{f^*}^n} D(\alpha) \quad \text{where} \quad D(\alpha) = \frac{1}{n}\sum_{i=1}^n -f_i^*(-\alpha_i) - \lambda g^*\left(\frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i - \frac{1}{\lambda}\psi\right), \tag{IV.4}$$

where for a function h, the Fenchel conjugate h^* is given by $h^*(v) = \sup_u uv - h(u)$, and $-\mathscr{D}_{f^*}$ is the domain of the function $x \mapsto \sum_{i=1}^n f_i^*(-x)$. This strategy is the one used by Stochastic Dual Coordinate Ascent (SDCA) [SSZ13]. The dual problem solutions are box-constrained to $-\mathscr{D}_{f^*}^n$ which is much easier to maintain than the open polytope $\Pi(X)$. Note that as all f_i are strictly decreasing (because they are strictly monotone on $(0, +\infty)$ with $\lim_{t\to 0} f_i(t) = +\infty$), their dual are defined on $\mathscr{D}_{f^*} \subset (-\infty, 0)$. By design, this approach keeps the dual constraints maintained all along the iterations and the following proposition, proved in Section 7.1, ensures that the primal iterate converges to a point of $\Pi(X)$.

Proposition 1. *Assume that the polytope $\Pi(X)$ is non-empty, the functions f_i are convex, differentiable, with $\lim_{t\to 0} f_i(t) = +\infty$ for $i = 1, \dots, n$ and that g is strongly convex. Then, strong duality holds, namely $P(w^*) = D(\alpha^*)$ and the Karush-Kuhn-Tucker conditions relate the two optima as*

$$\forall i \in \{1, \dots, n\},\ \alpha_i^* = -f_i^{*\prime}({w^*}^\top x_i) \quad \textit{and} \quad w^* = \nabla g^*\Big(\frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i - \frac{1}{\lambda}\psi\Big),$$

where $w^* \in \Pi(X)$ is the minimizer of P and $\alpha^* \in -\mathscr{D}_{f^*}^n$ the maximizer of D.

In this chapter, we introduce the log smoothness property and its characteristics and then we derive linear convergence rates for SDCA *without the gradient-Lipschitz* assumption, by replacing it with log smoothness, see Definition 1. Our results provide a state-of-the-art optimization technique for the considered problem (IV.2), with sound theoretical guarantees (see Section 3) and very strong empirical properties as illustrated on experiments conducted with several datasets for Poisson regression and Hawkes processes likelihood (see Section 5). We study also SDCA with importance sampling [ZZ15] under log smoothness and prove that it improves both theoretical guarantees and convergence rates observed in practical situations, see Sections 3.2 and 5. We provide also a heuristic initialization technique in Section 5.3 and a "mini-batch" [QRTF16] version of the algorithm in Section 5.4 that allows to end up with a particularly efficient solver for the considered problems. We motivate even further the problem considered in this chapter in Figure IV.1, where we consider a toy Poisson regression problem (with 2 features and 3 data points), for which L-BFGS-B typically fails while SDCA works. This illustrates the difficulty of the problem even on such an easy example.

Outline. We first introduce the log smoothness property in Section 2, relate it to self-concordance in Proposition 2 and translate it in the Fenchel conjugate space in Proposition 3. Then, we present the shifted SDCA algorithm in Section 3 and state its convergence guarantees in Theorem 1 under the log smoothness assumption. We also provide theoretical guarantees for variants of the algorithm, one using proximal

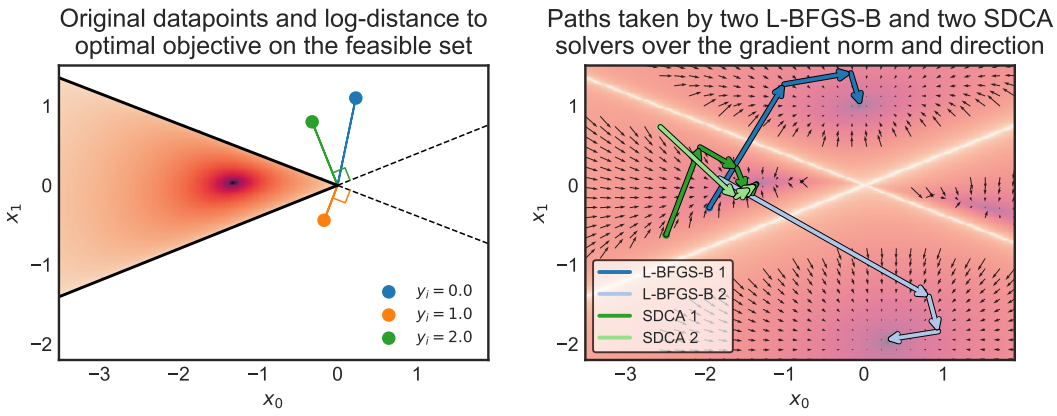

Figure IV.1: Iterates of SDCA and L-BFGS-B on a Poisson regression toy example with three samples and two features. *Left.* Dataset and value of the objective. *Right:* Iterates of L-BFGS-B and SDCA with two different starting points. The background represents the gradient norm and the arrows the gradient direction. SDCA is very stable and converges quickly towards the optimum, while L-BFGS-B easily converges out of the feasible space.

operators [SSZ14] and the second using importance sampling [ZZ15] which leads to better convergence guarantees in Theorem 2. In Section 4 we focus on two specific problems, namely Poisson regression and Hawkes processes, and explain how they fit into the considered setting. Section 5 contains experiments that illustrate the benefits of our approach compared to baselines. This Section also proposes a very efficient heuristic initialization and numerical details allowing to optimize over several indices at each iteration, which is a trick to accelerate even further the algorithm.

2 A tighter smoothness assumption

To have a better overview of what log smoothness is, we formulate the following proposition giving an equivalent property for log smooth functions that are twice differentiable.

Proposition 2. *Let $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ be a convex strictly monotone and twice differentiable function. Then,*

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x \in \mathscr{D}_f,\ f''(x) \leq \frac{1}{L} f'(x)^2.$$

This proposition is proved in Section 7.4 and we easily derive from it that $x \mapsto -L\log x$ on $(0, +\infty)$, $x \mapsto Lx$ on $\mathbb{R}$ and $x \mapsto L\exp(x)$ on $[0, +\infty)$ are L-log smooth.

This proposition is linked to the self-concordance property introduced by Nesterov [Nes13] widely used to study losses involving logarithms. For the sake of clarity, the results will be presented for functions whose domain $\mathscr{D}_f$ is a subset of $\mathbb{R}$ as this leads to lighter notations.

Definition 2. *A convex function $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ is standard self-concordant if*

$$\forall x \in \mathscr{D}_f,\ |f'''(x)| \leq 2f''(x)^{3/2}.$$

This property has been generalized [Bac10, STD17] but always consists in controlling the third order derivative by the second order derivative, initially to bound the second order Taylor approximation used in the Newton descent algorithm [Nes13]. While right hand sides of both properties ($f'(x)^2$ and $2f''(x)^{3/2}$) might look arbitrarily chosen, in fact they reflect the motivating use case of the logarithmic barriers where $f : t \mapsto -\log(t)$ and for which the bound is reached. Hence, log smoothness is the counterpart of self-concordance but to control the second order derivative with the first order derivative. As it is similarly built, log smoothness shares the affine invariant property with self-concordance. It means that if f_1 is L log-smooth then $f_2 : x \mapsto f_1(ax + b)$ with $a, b \in \mathbb{R}$ is also L-log smooth with the same constant L. An extension to the multivariate case where $\mathscr{D}_f \subset \mathbb{R}^d$ is likely feasible but is useless for our algorithm and hence beyond the scope of this paper. From the log smoothness property of a function f, we derive several characteristics for its Fenchel conjugate f^* starting with the following proposition.

Proposition 3. *Let $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ be a strictly monotone convex function and f^* be its twice differentiable Fenchel conjugate. Then,*

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \exists L > 0;\ \forall x \in \mathscr{D}_{f^*},\ f^{*\prime\prime}(x) \geq Lx^{-2}.$$

This proposition is proved in Section 7.5 and is the first step towards a series of convex inequalities for the Fenchel conjugate of log smooth functions. These inequalities, detailed in Section 7.6, bounds the Bregman divergence of such functions and are compared to what can be obtained with self-concordance or strong convexity (on a restricted set) in the canonical case where $f : t \mapsto -\log(t)$. It appears with log smoothness we obtain tighter bounds than what is achievable with other assumptions and that all these bounds are reached (and hence cannot be improved) in the canonical case (see Table IV.3).

3 The Shifted SDCA algorithm

The dual objective (IV.4) cannot be written as a composite sum of convex functions as in the general objective (IV.1), which is required for stochastic algorithms such as SRVG

[JZ13] or SAGA [DBLJ14]. It is better to use a coordinate-wise approach to optimize this problem, which leads to SDCA [SSZ14], in which the starting point has been *shifted* by $\frac{1}{\lambda}\psi$. This shift is induced by the relation linking primal and dual variables at the optimum: the second Karush-Kuhn-Tucker condition from Proposition 1,

$$w^* = \nabla g^*\Big(\frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^* x_i - \frac{1}{\lambda}\psi\Big). \tag{IV.5}$$

We first present the general algorithm (Algorithm IV.1), then its proximal alternative (Algorithm IV.2) and finally how importance sampling leads to better theoretical results. We assume that we know bounds $(\beta_i)_{1\le i\le n}$ such that $\beta_i/\alpha_i^* \ge 1$ for any $i = 1,\dots,n$, such bounds can be explicitly computed from the data in the particular cases considered in this chapter, see Section 4 for more details.

Algorithm IV.1 Shifted SDCA

Require: Bounds $\beta_i \in -\mathscr{D}_{f^*}$ such that $\forall i \in \{1,\dots,n\},\ \beta_i/\alpha_i^* \ge 1$,
$\alpha^{(0)} \in -\mathscr{D}_{f^*}^n$ dual starting point such that $\forall i \in \{1,\dots,n\},\ \beta_i/\alpha_i \ge 1$

$v^{(0)} = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^{(0)} x_i - \frac{1}{\lambda}\psi$
for $t = 1, 2 \dots T$ **do**
  Sample i uniformly at random in $\{1,\dots,n\}$
  Find α_i that maximizes $-\frac{1}{n}f_i^*(-\alpha_i) - \lambda g^*\big(v^{(t-1)} + (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i\big)$
  $\alpha_i \leftarrow \min(1, \beta_i/\alpha_i)\alpha_i$
  $\Delta\alpha_i \leftarrow \alpha_i - \alpha_i^{(t-1)}$
  $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \Delta\alpha_i e_i$
  $v^{(t)} \leftarrow v^{(t-1)} + (\lambda n)^{-1}\Delta\alpha_i x_i$
  $w^{(t)} \leftarrow \nabla g^*(v^{(t)})$
end for

The next theorem provides a linear convergence rate for Algorithm IV.1 where we assume that each f_i is L_i-log smooth (see Definition 1).

Theorem 1. *Suppose that we known bounds $\beta_i \in -\mathscr{D}_{f^*}$ such that $R_i = \frac{\beta_i}{\alpha_i^*} \ge 1$ for $i = 1,\dots,n$ and assume that all f_i are L_i-log smooth with differentiable Fenchel conjugates and g is 1-strongly convex. Then, Algorithm IV.1 satisfies*

$$\mathbb{E}[D(\alpha^*) - D(\alpha^{(t)})] \le \Big(1 - \frac{\min_i \sigma_i}{n}\Big)^t (D(\alpha^*) - D(\alpha^{(0)})), \tag{IV.6}$$

where

$$\sigma_i = \Big(1 + \frac{\|x_i\|^2 \alpha_i^{*2}}{2\lambda n L_i}\frac{(R_i-1)^2}{\frac{1}{R_i} + \log R_i - 1}\Big)^{-1}. \tag{IV.7}$$

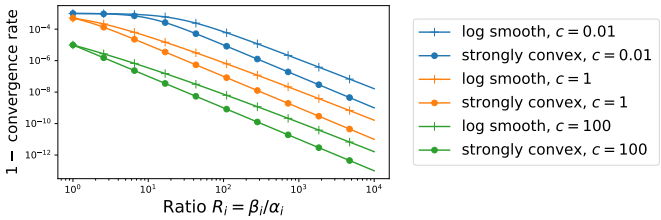

Figure IV.2: Evolution of one minus the convergence rate, namely σ_i/n for $n = 1000$ and different values of $c = (\|x_i\|^2 \alpha_i^{*2})/(\lambda n L_i)$ under log smoothness (this work) and strong convexity assumptions [SSZ13]. The convergence rate from Theorem 1 is one order of magnitude better.

The proof of Theorem 1 is given in Section 7.7. It states that in the considered setting, SDCA achieves a linear convergence rate for the dual objective. The bounds β_i are provided in Section 4 below for two particular cases: Poisson regression and likelihood Hawkes processes. Equipped with these bounds, we can compare the rate obtained in Theorem 1 with already known linear rates for SDCA under the gradient-Lipschitz assumption [SSZ13]. Indeed, we can restrict the domain of all f_i^* to $(-\beta_i, 0)$ on which Proposition 3 states that all f_i are $L_i/(\alpha_i^{*2} R_i^2)$-strongly convex. Now, following carefully the proof in [SSZ13] leads to the convergence rate given in Equation (IV.6) but with

$$\sigma_i = \left(1 + \frac{\|x_i\|^2 \alpha_i^{*2}}{\lambda n L_i} R_i^2\right)^{-1}.$$

Since $2\left(\frac{1}{R} + \log R - 1\right)(R-1)^{-2} \geq R^{-2}$ for any $R \geq 1$, Theorem 1 provides a faster convergence rate. The comparative gain depends on the values of $(\|x_i\|^2 \alpha_i^{*2})/(\lambda n L_i)$ and R_i but it increases logarithmically with the value of R_i. Table IV.1 below compares the explicit values of these linear rates on a dataset used in our experiments for Poisson regression and Figure IV.2 shows that the rate obtained under log smoothness assumption is one order of magnitude better than the one obtained with strong convexity.

Remark 1. *Convergence rates for the primal objective are not provided since the primal iterate $w^{(t)}$ typically belongs to $\Pi(X)$ only when it is close enough to the optimum. This would make most of the values of the primal objective $P(w^{(t)})$ undefined and therefore not comparable to $P(w^*)$.*

3.1 Proximal algorithm

Algorithm IV.1 maximizes the dual over one coordinate at Line 4 whose solution might not be explicit and requires inner steps to obtain $\alpha_i^{(t)}$. But, whenever g can be

written as

$$g(w) = \tfrac{1}{2}\|w\|^2 + h(w), \tag{IV.8}$$

where h is a convex, prox capable and possibly non-differentiable function, we use the same technique as Prox-SDCA [SSZ14] with a proximal lower bound that leads to

$$\alpha_i^{(t)} = \underset{a_i \in -\mathscr{D}_{f^*}}{\arg\max} -f_i^*(-\alpha_i) - \frac{\lambda n}{2}\left\| w^{(t-1)} - (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i \right\|^2,$$

with

$$w^{(t-1)} = \mathrm{prox}_h\Big(\frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^{(t-1)} x_i - \frac{1}{\lambda}\psi\Big),$$

see Section 7.2 for details. This leads to a proximal variant described in Algorithm IV.2 below, which is able to handle various regularization techniques and which has the same convergence guarantees as Algorithm IV.1 given in Theorem 1. Also, note that assuming that g can be written as (IV.8) with a prox-capable function h is rather unrestrictive, since one can always add a ridge penalization term in the objective.

Algorithm IV.2 Shifted Prox-SDCA

Require: Bounds $\beta_i \in -\mathscr{D}_{f^*}$ such that $\forall i \in \{1,\dots,n\},\ \beta_i/\alpha_i^* \geq 1$,
$\alpha^{(0)} \in -\mathscr{D}_{f^*}^n$ dual starting point such that $\forall i \in \{1,\dots,n\},\ \beta_i/\alpha_i \geq 1$

1: $v^{(0)} = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^{(0)} x_i - \frac{1}{\lambda}\psi$
2: **for** $t = 1, 2 \dots T$ **do**
3: Sample i uniformly at random in $\{1,\dots,n\}$
4: Find α_i that maximize $-\frac{1}{n}f_i^*(-\alpha_i) - \frac{\lambda}{2}\left\| w^{(t-1)} + (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i \right\|^2$
5: $\alpha_i \leftarrow \min(1, \beta_i/\alpha_i)\alpha_i$
6: $\Delta\alpha_i \leftarrow \alpha_i - \alpha_i^{(t-1)}$
7: $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \Delta\alpha_i e_i$
8: $v^{(t)} \leftarrow v^{(t-1)} + (\lambda n)^{-1}\Delta\alpha_i x_i$
9: $w^{(t)} \leftarrow \mathrm{prox}_h(v^{(t)})$
10: **end for**

3.2 Importance sampling

Importance sampling consists in adapting the probabilities of choosing a sample i (which is by default done uniformly at random, see Line 3 from Algorithm IV.1) using the improvement which is expected by sampling it. Consider a distribution ρ on $\{1,\dots,n\}$ with probabilities $\{\rho_1,\dots,\rho_n\}$ such that $\rho_i \geq 0$ for any i and $\sum_{i=1}^n \rho_i = 1$. The

Shifted SDCA and Shifted Prox-SDCA with importance sampling algorithms are simply obtained by modifying the way i is sampled in Line 3 of Algorithms IV.1 and IV.2: instead of sampling uniformly at random, we sample using such a distribution ρ. The optimal sampling probability ρ is obtained in the same way as [ZZ15] and it also leads under our log smoothness assumption to a tighter convergence rate, as stated in Theorem 2 below.

Theorem 2. *Suppose that we known bounds $\beta_i \in -\mathscr{D}_{f^*}$ such that $R_i = \frac{\beta_i}{\alpha_i^*} \geq 1$ for $i = 1,\dots,n$ and assume that all f_i are L_i-*log *smooth with differentiable Fenchel conjugates and g is 1-strongly convex. Consider σ defined by* (IV.7) *and consider the distribution*

$$\rho_i = \frac{\sigma_i^{-1}}{\sum_{j=1}^n \sigma_j^{-1}}$$

for $i \in \{1,\dots,n\}$. Then, Algorithm IV.1 and IV.2 where Line 3 is replaced by sampling $i \sim \rho$ satisfy

$$\mathbb{E}[D(\alpha^*) - D(\alpha^{(t)})] \leq \left(1 - \frac{\bar{\sigma}}{n}\right)^t (D(\alpha^*) - D(\alpha^{(0)}),$$

where $\bar{\sigma} = \left(\frac{1}{n}\sum_{i=1}^n \sigma_i^{-1}\right)^{-1}$.

The proof is given in Section 7.8. This convergence rate is stronger than the previous one from Theorem 1 since $(\frac{1}{n}\sum_{i=1}^n \sigma_i^{-1})^{-1} \geq \min_i \sigma_i$. Table IV.1 below compares the explicit values of these linear rates on a dataset used in our experiments for Poisson regression (facebook dataset). We observe that the log smooth rate with importance sampling is orders of magnitude better than the one obtained with the standard theory for SDCA which exploits only the $L_i/(\alpha_i^{*2} R_i^2)$ strong convexity of the functions f_i^*.

strongly convex	strongly convex with importance sampling	log smooth	log smooth with importance sampling
$(0.9999)^t$	$(0.9969)^t$	$(0.9984)^t$	$(\mathbf{0.9679})^t$

Table IV.1: Theoretical convergence rates obtained on the facebook dataset (see Section 5.1) in four different settings: strongly convex (which is the rate obtained when all functions f_i are considered $L_i/(\alpha_i^{*2} R_i^2)$-strongly convex) with and without importance sampling [SSZ13, ZZ15] and the rate obtained in the setting considered in the chapter, with and without importance sampling. In this experiment, the maximum value for R_i is 9062 and its average value is 308. As expected, the best rate is obtained by combining the log-smoothness property with importance sampling.

4 Applications to Poisson regression and Hawkes processes

In this Section we describe two important models that fit into the setting of this chapter. We precisely formulate them as in Equation (IV.2) and give the explicit value of bounds β_i such as $\alpha_i^* \leq \beta_i$, where α^* is the solution to the dual problem (IV.4).

4.1 Linear Poisson regression

Poisson regression is widely used to model count data, namely when, in the dataset, each observation $x_i \in \mathbb{R}^d$ is associated an integer output $y_i \in \mathbb{N}$ (the set of non negative integers) for $i = 1, \dots, n$. It aims to find a vector $w \in \mathbb{R}^d$ such that for a given function $\phi : \mathscr{D}_\phi \subset \mathbb{R} \to (0, +\infty)^+$, y_i is the realization of a Poisson random variable of mean $\phi(w^\top x_i)$. A convenient choice is to use exp for ϕ as it always guarantees that $\phi(w^\top x_i) > 0$. However, using the exponential function assumes that the covariates have a multiplicative effect that often cannot be justified. The tougher problem of linear Poisson regression, where $\phi(t) = t$ and $\mathscr{D}_\phi$ is the polytope $\Pi(X)$, appears to model additive effects. For example, this applies in image reconstruction. The original image is retrieved from photons counts y_i distributed as a Poisson distribution with intensity $w^\top x_i$, that are received while observing the image with different detectors represented by the vectors $x_i \in \mathbb{R}^d$. This application has been extensively studied in the literature, see [HMW12, BB16, TDKC15] and [BBDV09] for a review with a hundred references. Linear Poisson regression is also used in various fields such as survival analysis with additive effects [BF10] and web-marketing [CPC09] where the intensity corresponds to an intensity of clicks on banners in web-marketing. To formalize, we consider a training dataset $(x_1, y_1), \dots, (x_{n_0}, y_{n_0})$ with $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{N}$ and assume without loss of generality that $y_i > 0$ for $i \in \{1, \dots, n\}$ while $y_i = 0$ for $i \in \{n+1, \dots, n_0\}$ where $n = \#\{i : y_i > 0\} \leq n_0$ (this simply means that we put first the samples corresponding to a label $y_i > 0$). The negative log-likelihood of this model with a penalization function g can be written as

$$P_0(w) = \frac{1}{n_0} \sum_{i=1}^{n_0} (w^\top x_i - y_i \log(w^\top x_i)) + \lambda_0 g(w)$$

where $\lambda_0 > 0$ corresponds to the level of penalization, with the constraint that $w^\top x_i$ for $i = 1, \dots, n$. This corresponds to Equation (IV.2) with $f_i(w) = -y_i \log(w^\top x_i)$ for $i = 1, \dots, n$, which are y_i-log smooth functions, and with

$$\psi = \frac{1}{n} \sum_{i=1}^{n_0} x_i \quad \text{and} \quad \lambda = \frac{n_0}{n} \lambda_0.$$

Note that the zero labeled observations can be safely removed from the sum and are fully encompassed in ψ. The algorithms and results proposed in Section 3 can therefore be applied for this model.

4.2 Hawkes processes

Hawkes processes are used to study cross causality that might occur in one or several events series. First, they were introduced to study earthquake propagation, the network across which the aftershocks propagate can be recovered given all tremors timestamps [Oga99]. Then, they have been used in high frequency finance to describe market reactions to different types of orders [BMM15]. In the recent years Hawkes processes have found many new applications including crime prediction [Moh13] or social network information propagation [LSV+16]. A Hawkes process [HO74] is a multivariate point-process: it models timestamps $\{t_k^i\}_{i\geq 1}$ of nodes $i = 1,\dots,I$ using a multivariate counting process with a particular auto-regressive structure in its intensity. More precisely, we say that a multivariate counting process $N_t = [N_t^1,\dots,N_t^I]$ where $N_t^i = \sum_{k\geq 1} \mathbb{1}_{t_k^i \leq t}$ for $t \geq 0$ is a Hawkes process if the intensity of N^i has the following structure:

$$\lambda^i(t) = \mu_i + \sum_{j=1}^{I} \int \phi_{ij}(t-s)\,\mathrm{d}N^j(s) = \mu_i + \sum_{j=1}^{I} \sum_{k\,:\,t_k^j < t} \phi_{ij}(t - t_k^j).$$

The $\mu_i \geq 0$ are called *baselines* intensities, and correspond to the exogenous intensity of events from node i, and the functions ϕ_{ij} for $1 \leq i,j \leq I$ are called *kernels*. They quantify the influence of past events from node j on the intensity of events from node i. The main parametric model for the kernels is the so-called *exponential* kernel, in which we consider

$$\phi^{ij}(t) = \sum_{u=1}^{U} a_u^{ij} b_u \exp(-b_u t) \tag{IV.9}$$

with $b_u > 0$. In this model the matrix $A = [\sum_{u=1}^{U} a_u^{ij}]_{1\leq i,j\leq d}$ is understood as an *adjacency matrix*, since entry $A_{i,j}$ quantifies the impact of the activity of node j on the activity of node i, while $b_u > 0$ are memory parameters. We stack these parameters into a vector θ containing the baselines μ_i and the self and cross-excitation parameters a_u^{ij}. Note that in this model the memory parameters b_u are supposed to be given. The associated goodness-of-fit is the negative log-likelihood, which is given by the general theory of point processes (see [DVJ07]) as

$$-\ell(\theta) = -\sum_{i=1}^{I} \ell_i(\theta), \quad \text{with} \quad -\ell_i(\theta) = \int_0^T \lambda_\theta^i(t)\,\mathrm{d}t - \int_0^T \log(\lambda_\theta^i(t))\,\mathrm{d}N^i(t).$$

Let us define the following weights for $i, j = 1, \dots I$ and $u = 1, \dots, U$,

$$g_u^j(t) = \sum_{k \,:\, t_k^j < t} b_u e^{-b_u (t - t_k^j)}, \quad g_{u,k}^{ij} = g_u^j(t_k^i) \quad \text{and} \quad G_u^j = \int_0^T g_u^j(t) \, \mathrm{d}t \tag{IV.10}$$

that can be computed efficiently for exponential kernels thanks to recurrence formulas (the complexity is linear with respect to the number of events of each node). Using the parametrization of the kernels from Equation (IV.9) we can rewrite each term of the negative log-likelihood as

$$-\ell_i(\mu_i, a^i) = \sum_{i=1}^{I} \Big[\mu^i T + \sum_{j=1}^{I} \sum_{u=1}^{U} a_u^{ij} G_u^j - \sum_{k=1}^{n_i} \log\Big(\mu^i + \sum_{j=1}^{I} \sum_{u=1}^{U} a_u^{ij} g_{u,k}^{ij} \Big) \Big].$$

To rewrite ℓ_i in a vectorial form we define n_i as the number of events of node i and the following vectors for $i = 1, \dots, I$:

$$w^i = \begin{bmatrix} \mu^i & a_1^{i,1} & \cdots & a_U^{i,1} & \cdots & a_1^{i,I} & \cdots & a_U^{i,I} \end{bmatrix}^\top,$$

that are the model weights involved in ℓ_i, and

$$\psi^i = \frac{1}{n_i} \begin{bmatrix} T & G_1^1 & \cdots G_U^1 & \cdots & G_1^I & \cdots & G_U^I \end{bmatrix}^\top,$$

which correspond to the vector involved in the linear part of the primal objective (IV.2) and finally

$$x_k^i = \begin{bmatrix} 1 & g_{1,k}^{i,1} & \cdots & g_{U,k}^{i,1} & \cdots & g_{1,k}^{i,I} & \cdots & g_{U,k}^{i,I} \end{bmatrix}^\top,$$

for $k = 1, \dots, n_i$ which contains all the timestamps data computed in the weights computed in Equation (IV.10). With these notations the negative log-likelihood for node i can be written as

$$-\ell(w) = -\sum_{i=1}^{I} \ell_i(w^i) \quad \text{with} \quad -\tfrac{1}{n_i} \ell_i(w^i) = (w^i)^\top \psi^i - \frac{1}{n_i} \sum_{k=1}^{n_i} \log((w^i)^\top x_k^i).$$

First, it shows that the negative log-likelihood can be separated into I independent sub-problems with goodness-of-fit $-\ell_i(w^i)$ that corresponds to the intensity of node i with the weights $x_{i,k}$ carrying data from the events of the other nodes j. Each subproblem is a particular case of the primal objective (IV.2), where all the labels y_i are equal to 1. As a consequence, we can use the algorithms and results from Section 3 to train penalized multivariate Hawkes processes very efficiently.

4.3 Closed form solution and bounds on dual variables

In this Section with provide the explicit solution to Line 4 of Algorithm IV.2 when the objective corresponds to the linear Poisson regression or the Hawkes process goodness-of-fit. In Proposition 4 below we provide the closed-form solution of the local maximization step corresponding to Line 4 of Algorithm IV.2.

Proposition 4. *For Poisson regression and Hawkes processes, Line 4 of Algorithm IV.2 has a closed form solution, namely*

$$\alpha_i^t = \frac{1}{2}\left(\sqrt{\left(\alpha_i^{(t-1)} - \frac{\lambda n}{\|x_i\|^2} w^{(t-1)\top} x_i\right)^2 + 4\lambda n \frac{y_i}{\|x_i\|^2}} + \alpha_i^{(t-1)} - \frac{\lambda n}{\|x_i\|^2} w^{(t-1)\top} x_i\right).$$

This closed-form expression allows to derive a numerically very efficient training algorithm, as illustrated in Section 5 below. For these two use cases, the dual loss is given by $f_i^*(-\alpha_i) = -y_i - y_i \log(\frac{\alpha_i}{y_i})$ for any $\alpha_i > 0$ (with $y_i = 1$ for the Hawkes processes). For this specific dual loss, we can provide also upper bounds β_i for all optimal dual variables α_i^*, as stated in the next Proposition.

Proposition 5. *For Poisson regression and Hawkes processes, if* $g(w) = \frac{1}{2}\|w\|^2$ *and if* ${x_i}^\top x_j \geq 0$ *for all* $1 \leq i, j \leq n$*, we have the following upper bounds on the dual variables at the optimum*:

$$\alpha_i^* \leq \beta_i \quad \textit{where} \quad \beta_i = \frac{1}{2\|x_i\|^2}\left(n\psi^\top x_i + \sqrt{(n\psi^\top x_i)^2 + 4\lambda n y_i \|x_i\|^2}\right)$$

for any $i = 1, \ldots, n$.

The proofs of Propositions 4 and 5 are provided in Section 7.9. Note that the inner product assumption ${x_i}^\top x_j \geq 0$ from Proposition 5 is mild: it is always met for the Hawkes process with kernels given by (IV.9) and it it met for Poisson regression whenever one applies for instance a min-max scaling on the features matrix.

Remark 2. *The closed form solution from Proposition 4 is always lower than the generic bound* β_i*, as explained in Section 7.11. Hence, we actually do not need to manually bound* $\alpha_i^{(t)}$ *at line 5 of Algorithm IV.1 in this particular case.*

5 Experiments

To evaluate efficiently Shifted SDCA we have compared it with other optimization algorithms that can handle the primal problem (IV.2) nicely, without the gradient-Lipschitz assumptions. We have discarded the modified proximal gradient method from [TDKC15] since most of the time it was diverging while computing the initial step with the Barzilai-Borwein method on the considered examples. We consider the following algorithms.

NoLips. This is a first order batch algorithm that relies on relative-smoothness [LFN18] instead of the gradient-Lipschitz assumption. Its application to linear Poisson regression has been detailed in [BB16] and its analysis provides convergence guarantees with a sublinear convergence rate in $\mathcal{O}(1/n)$. However, this method is by design limited to solutions with positive entries (namely $w^* \in [0,\infty)^d$) and provides guarantees only in this case. Its theoretical step-size decreases linearly with $1/n$ and is too small in practice. Hence, we have tuned the step-size to get the best objective after 300 iterations.

SVRG. This is a stochastic gradient descent algorithm with variance reduction introduced in [JZ13, XZ14]. We used a variant introduced in [TMDQ16], which uses Barzilai-Borwein in order to adapt the step-size, since gradient-Lipschitz constants are unavailable in the considered setting. We consider this version of variance reduction, since alternatives such as SAGA[DBLJ14] and SAG [SLRB17] do not propose variants with Barzilai-Borwein type of step-size selection.

L-BFGS-B. L-BFGS-B is a limited-memory quasi-Newton algorithm [Noc80, NW06]. It relies on an estimation of the inverse of the Hessian based on gradients differences. This technique allows L-BFGS-B to consider the curvature information leading to faster convergence than other batch first order algorithms such as ISTA and FISTA [BT09].

Newton algorithm. This is the standard second-order Newton algorithm which computes at each iteration the hessian of the objective to solve a linear system with it. In our experiments, the considered objectives are both log-smooth and self-concordant [Nes13]. The self-concordant property bounds the third order derivative by the second order derivative, giving explicit control of the second order Taylor expansion [Bac10]. This ensures supra-linear convergence guarantees and keeps all iterates in the open polytope (IV.3) if the starting point is in it [NN94]. However, the computational cost of the hessian inversion makes this algorithm scale very poorly with the number of dimensions d (the size of the vectors x_i).

SDCA. This is the Shifted-SDCA algorithm, see Algorithm IV.2, without importance sampling. Indeed, the bounds given in Proposition 5 are not tight enough to improve convergence when used for importance sampling in the practical situations considered in this Section (despite the fact that the rates are theoretically better). A similar behavior was observed in [PTLJ18].

SVRG and L-BFGS-B are almost always diverging in these experiments just like in the simple example considered in Figure IV.1. Hence, the problems are tuned to avoid any violation of the open polytope constraint (IV.3), and to output comparable results

Table IV.2: Poisson datasets details.

dataset	wine [2]	facebook [3]	vegas [4]	news [5]	property [6]	simulated [7]
# lines	4898	500	2215	504	50099	100000
# features	11	41	102	160	194	100

between algorithms. Namely, to ensure that $w^\top x_i > 0$ for any iterate w, we scale the vectors x_i so that they contain only non-negative entries, and the iterates of SVRG and L-BFGS-B are projected onto $[0,+\infty)^d$. This highlights two first drawbacks of these algorithms: they cannot deal with a generic feature matrix and their solutions contain only non-negative coefficients. For each run, the simply take $\lambda = \overline{x}/n$ where $\overline{x} = \frac{1}{n}\sum_{i=1}^n \|x_i\|^2$. This simple choice seemed relevant for all the considered problems.

5.1 Poisson regression

For Poisson regression we have processed our feature matrices to obtain coefficients between 0 and 1. Numerical features are transformed with a min-max scaler and categorical features are one hot encoded. We run our experiments on six datasets found on UCI dataset repository [Lic13] and Kaggle[1] (see Table IV.2 for more details). These datasets are used to predict a number of interactions for a social post (news and facebook), the rating of a wine or a hotel (wine and vegas) or the number of hazards occurring in a property (property). The last one comes from simulated data which follows a Poisson regression. In Figure IV.3 we present the convergence speed of the five algorithms. As our algorithms follow quite different schemes, we measure this speed regarding to the computational time. In all runs, NoLips, SVRG and L-BFGS-B cannot reach the optimal solution as the problem minimizer contains negative values. This is illustrated in detail in Figure IV.4 for vegas dataset where it appears that all solvers obtain similar results for the positive values of w^* but only Newton and SDCA algorithms are able to estimate the negatives values of w^*. As expected, the Newton algorithm becomes very slow as the number of features d increases. SDCA is the only first order solver that reaches the optimal solution. It combines the best of both world, the scalability of a first order solver and the ability to reach solutions with negative entries.

5.2 Hawkes processes

If the adjacency matrix A is forced to be entrywise positive, then no event type can have an inhibitive effect on another. This ability to exhibit inhibitive effect has direct

[1] https://www.kaggle.com/datasets

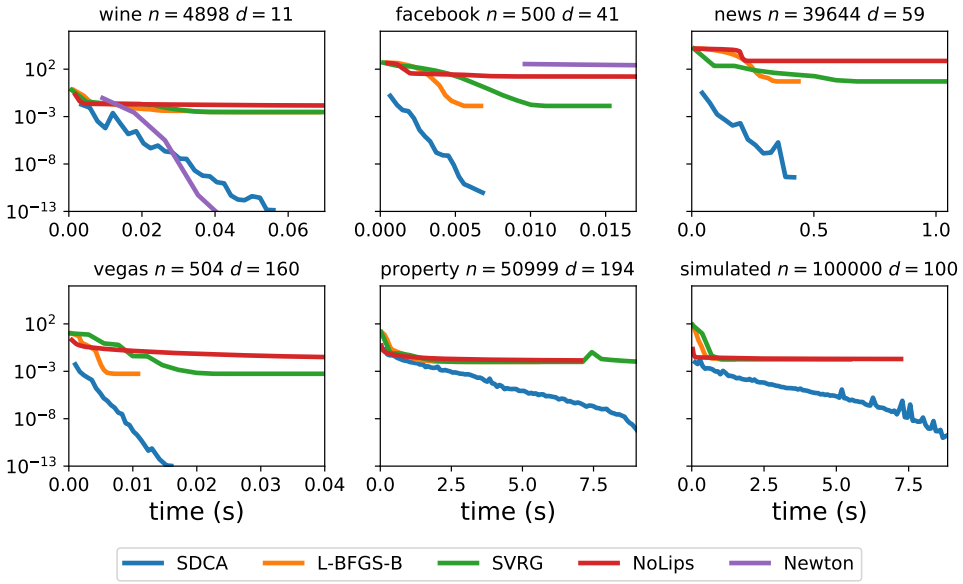

Figure IV.3: Convergence over time of five algorithms SDCA, SVRG, NoLips, L-BFGS-B and Newton on 6 datasets of Poisson regression. SDCA combines the best of both worlds: speed and scalability of SVRG and L-BFGS-B with the precision of Newton's solution.

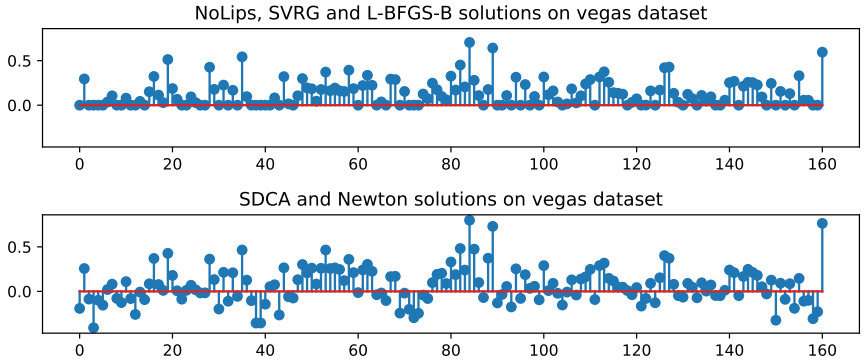

Figure IV.4: Estimated minimizers w^* on the vegas dataset (160 features). The positive entries are roughly similarly recovered by all solvers but the negative entries are only retrieved by SDCA and Newton algorithms.

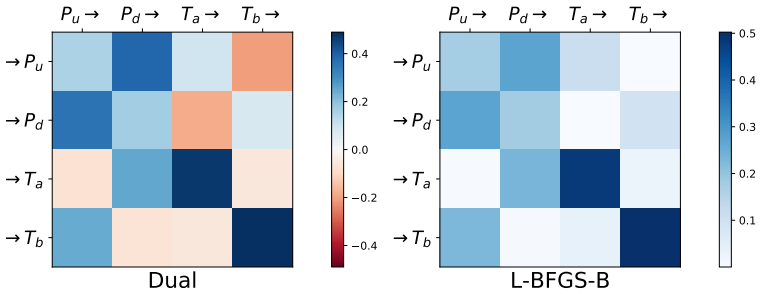

Figure IV.5: Adjacency matrix A of a Hawkes process fitted on high-frequency financial data from the Bund market. This reproduces an experiment run in [BJM16] where P_u (resp. P_d) counts the number of upward (resp. downward) mid-price moves and T_a (resp. T_b) counts the number of market orders at the best ask (resp. best bid) that do not move the price. SDCA detects inhibitive behaviors while L-BFGS-B cannot.

implications on real life datasets especially in finance where these effects are common [BJM16, BMM15, RBL17]. In Figure IV.5 we present the aggregated influence of the kernels obtained after training a Hawkes process on a finance dataset exploring market microstructure [BMM15]. While L-BFGS-B (or SVRG, or NoLips) recovers only excitation in the adjacency matrix, SDCA also retrieves inhibition that one event type might have on another. It is expected that when stocks are sold (resp. bought) the price is unlikely to go up (resp. down) but this is retrieved by SDCA only. On simulated data this is even clearer and in Figure IV.6 we observe the same behavior when the ground truth contains inhibitive effects. Our experiment consists in two simulated Hawkes processes with 10 nodes and sum-exponential kernels with 3 decays. There are only excitation effects - all a_u^{ij} are positive - in the first case and we allow inhibitive effects in the second. Events are simulated according to these kernels that we try to recover. While it would be standard to compare the performances in terms of log-likelihood obtained on the a test sample, nothing ensures that the problem optimizer lies in the feasible set of the test set. Hence the results are compared by looking at the estimation error (RMSE) of the adjacency matrix A across iterations. Figure IV.6 shows that SDCA always converges faster towards its solution in both cases and that when the adjacency matrix contains inhibitive effects, SDCA obtains a better estimation error than L-BFGS-B.

5.3 Heuristic initialization

The default dual initialization in [SSZ14] ($\alpha^{(0)} = 0_n$) is not a feasible dual point. Instead of setting arbitrarily $\alpha^{(0)}$ to 1_n, we design, from three properties, a vector $\kappa \in -\mathscr{D}_{f^*}^n$

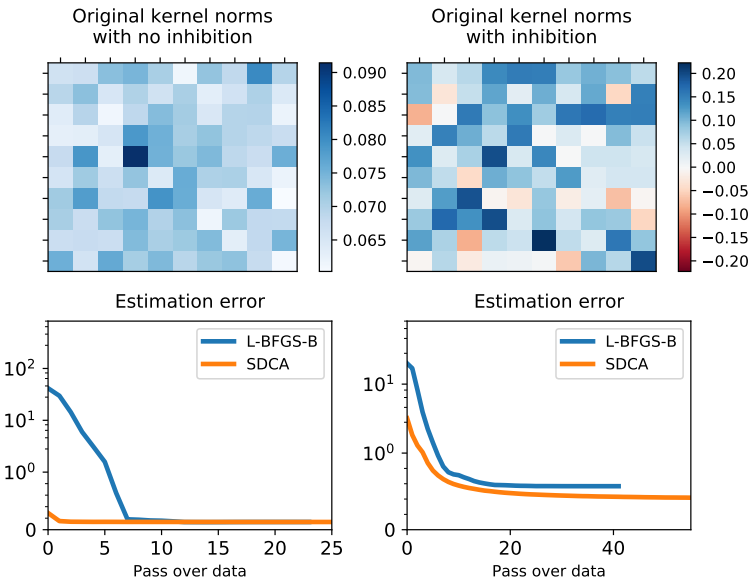

Figure IV.6: *Top:* Adjacency matrix of the Hawkes processes used for simulation. *Bottom:* estimation error of the adjacency matrix A across iterations. In both cases SDCA is faster than L-BFGS-B and reaches a better estimation error when there are inhibitive effects to recover (*Right*).

that is linearly linked to α^* and then rely on Proposition 7 to find a heuristic starting point $\alpha^{(0)}$ from κ for Poisson regression and Hawkes processes.

Property 1: link with $\|x_i\|$. Proposition 6 relates exactly α_i^* to the inverse of the norm of x_i.

Proposition 6. *For Poisson regression and Hawkes processes, the value of the dual optimum α_i^* is linearly linked to the inverse of the norm of x_i. Namely, if there is $c_i > 0$ such that $\xi_i = c_i x_i$ for any $i \in \{1, \dots n\}$, then ζ^*, the solution of the dual problem*

$$\operatorname*{argmax}_{\zeta \in (0,+\infty)^n} \frac{1}{n} \sum_{i=1}^{n} y_i + y_i \log\left(\frac{\zeta_i}{y_i}\right) - \lambda g^*\left(\frac{1}{\lambda n} \sum_{i=1}^{n} \zeta_i \xi_i - \frac{1}{\lambda}\psi\right),$$

satisfies $\zeta_i^* = \alpha_i^* / c_i$ for all $i = 1, \dots n$.

This Proposition is proved in Section 7.12. It suggests to consider $\kappa_i \propto 1/\|x_i\|$ for all $i = 1, \dots n$.

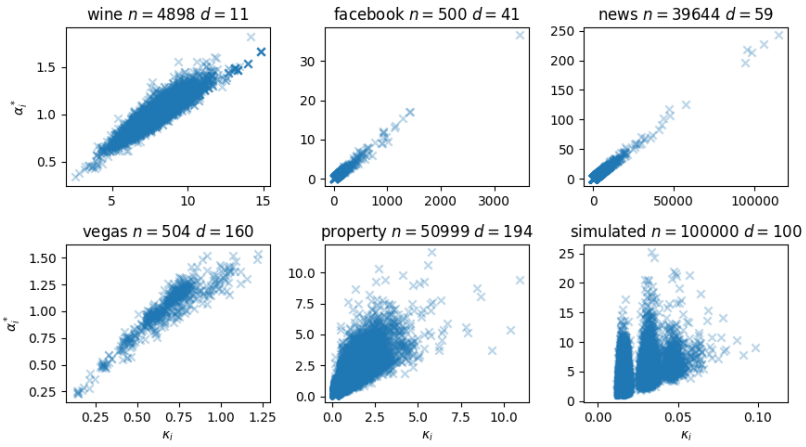

Figure IV.7: Value of α_i^* given κ_i for $i = 1, \dots, n$. There is a linear link relating initial guess κ_i to the dual optimum α_i^* on Poisson datasets but the amplitude is not adjusted yet.

Property 2: link with y_i. For Poisson regression and Hawkes processes where $f_i(x) = -y_i \log x$, the second Karush-Kuhn-Tucker Condition (IV.14) (see Section 7.1 for more details) writes

$$\alpha_i^* = \frac{y_i}{w^{*\top} x_i}$$

for $i = 1, \dots, n$. Hence, α^* and y are correlated (a change in y_i only leads to a minor change in w^*), so we will consider $\kappa_i \propto y_i / \|x_i\|$.

Property 3: link with the features matrix. The inner product $w^{*\top} x_i$ is positive and at the optimum, the Karush-Kuhn-Tucker Condition (IV.5) (which links w^* to x_i through α_i^*) tells that α_i^* is likely to be large if x_i is poorly correlated to other features, i.e. if $x_i^\top \sum_{j=1}^n x_j$ is small. Finally, the choice

$$\kappa_i = \frac{y_i}{x_i^\top \sum_{j=1}^n x_j} \tag{IV.11}$$

for $i = 1, \dots, n$, takes these three properties into account.

Figure IV.7 plots the optimal dual variables α^* from the Poisson regression experiments of Section 5.1 against the the κ vector from Equation IV.11. We observe in these experiments a good correlation between the two, but κ is only a good guess for initialization $\alpha^{(0)}$ up to a multiplicative factor that the following proposition aims to find.

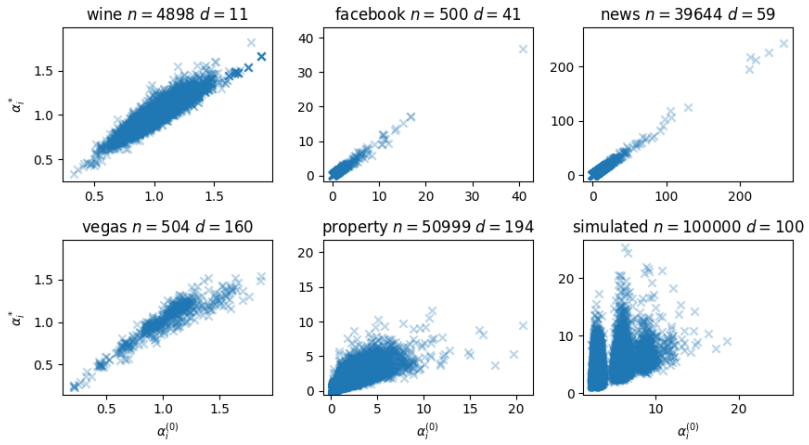

Figure IV.8: Value of α_i^* given $\alpha_i^{(0)}$ from Equation (IV.12) for $i = 1, \dots, n$. These values are close an correlated which makes $\alpha^{(0)}$ a good initialization value.

Proposition 7. *For Poisson regression and Hawkes processes and* $g(w) = \frac{1}{2}\|w\|^2$, *if we constrain the dual solution* $\alpha^* \in (0, +\infty)^n$ *to be collinear with a given vector* $\kappa \in (0, +\infty)^n$, *i.e.* $\alpha^* = \bar{\alpha}\kappa$ *for some* $\bar{\alpha} \in \mathbb{R}$, *then the optimal value for* $\bar{\alpha}$ *is given by*

$$\bar{\alpha} = \frac{\psi^\top \chi_\kappa + \sqrt{(\psi^\top \chi_\kappa)^2 + 4\lambda \|\chi_\kappa\|^2 \frac{1}{n}\sum_{i=1}^n y_i}}{2\|\chi_\kappa\|^2} \quad \textit{with} \quad \chi_\kappa = \frac{1}{n}\sum_{i=1}^n \kappa_i x_i.$$

Combined with the previous properties, we suggest to consider

$$\alpha_i^{(0)} = \bar{\alpha}\kappa_i \tag{IV.12}$$

as an initial point, where κ_i *is defined in Equation* (IV.11).

This Proposition is proved in Section 7.13. Figure IV.8 presents the values of α_i^* given its initial value $\alpha_i^{(0)}$ for $i = 1, \dots, n$ and shows that the rescaling has worked properly. We validate this heuristic initialization by showing that it leads to a much faster convergence in Figure IV.9 below. Indeed, we observe that SDCA initialized with Equation (IV.12) reaches optimal objective much faster than when initialization consists in setting all dual variables arbitrarily to 1.

5.4 Using mini batches

At each step t, SDCA [SSZ13] maximizes the dual objective by picking one index $i_t \in \{1, \dots, n\}$ and maximizing the dual objective over the coordinate i_t of the dual

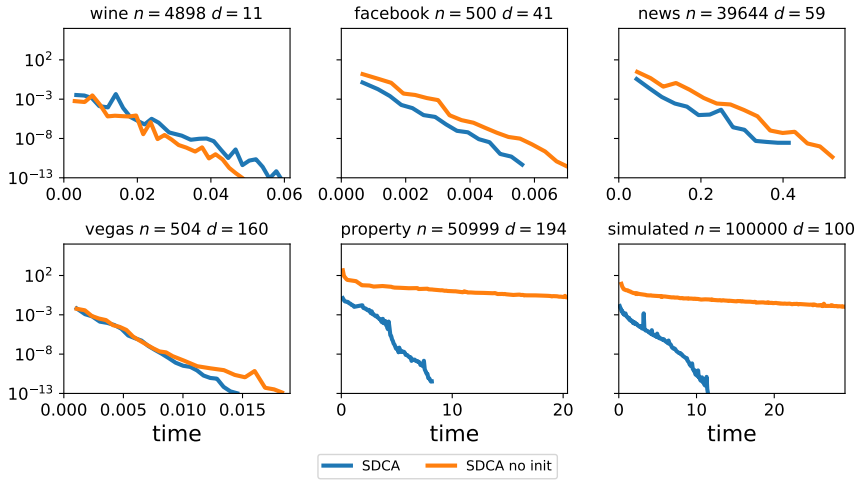

Figure IV.9: Convergence over time of SDCA with wise initialization from Equation (IV.12) and SDCA arbitrarily initialized with $\alpha^{(0)} = 1$.

vector α, and sets

$$\alpha_i^{t+1} = \underset{v \in -\mathscr{D}_{f^*}}{\operatorname{argmax}} D(\alpha_1^t, \ldots, \alpha_{i_t-1}^t, v, \alpha_{i_t+1}^t, \ldots, \alpha_n).$$

In some cases this maximization has a closed-form solution, such as for Poisson regression (see Proposition 4) or least-squares regression where $f_i(w) = (y_i - w^\top x_i)^2$ leads to the explicit update

$$\alpha_i^{t+1} = \alpha_i^t + \frac{y_i - w^\top x_i - \alpha_i^t}{1 + (\lambda n)^{-1} \|x_i\|^2}.$$

In some other cases, such as logistic regression, this closed form solution cannot be exhibited and we must perform a Newton descent algorithm. Each Newton step consists in computing $\partial D(\alpha)/\partial \alpha_i$ and $\partial^2 D(\alpha)/\partial \alpha_i^2$, which are one dimensional operations given $\|x_i\|^2$ and $w^\top x_i$. Hence, in a large dimensional setting, when the observations x_i have many non zero entries, the main cost of the steps resides mostly in computing $\|x_i\|^2$ and $w^\top x_i$. Since $\|x_i\|^2$ and $w^\top x_i$ must also be computed when using a closed-form solution, using Newton steps instead of the closed-form is eventually not much more computationally expensive. So, in order to obtain a better trade-off between Newton steps and inner-products computations, we can consider more than a single index on which we maximize the dual objective. This is called the mini-batch approach, see Stochastic Dual Newton Ascent (SDNA) [QRTF16]. It consists in selecting a set $\mathscr{I} \subset \{1, \ldots, n\}$ of p indices at each iteration t. The value of

α_i^{t+1} becomes in this case

$$\alpha_i^{t+1} = \underset{v\in(-\mathscr{D}_{f^*})^p}{\operatorname{argmax}} D(b_1,\dots,b_n) \quad \text{where} \quad b_i = \begin{cases} v_j & \text{if} \quad i\in\mathscr{I} \text{ and } j \text{ is the position of } i \text{ in } \mathscr{I} \\ \alpha_i^t & \text{otherwise.} \end{cases}$$

The two extreme cases are $p=1$, which is the standard SDCA algorithm, and $p=n$ for which we perform a full Newton algorithm. After computing the inner products $w^\top x_i$ and $x_i^\top x_j$ for all $(i,j)\in\mathscr{I}^2$ each iteration will simply performs up to 10 Newton steps in which the bottleneck is to solve a $p\times p$ linear system. This allows to better exploit curvature and obtain better convergence guarantees for gradient-Lipschitz losses [QRTF16].

We can apply this to Poisson regression and Hawkes processes where $f_i(x) = -y_i\log x$. The maximization steps of Line 4 in Algorithm IV.2 is now performed on a set of coordinates $\mathscr{I}\subset\{1,\dots n\}$ and consists in finding

$$\max_{\alpha_i;i\in\mathscr{I}} D_{\mathscr{I}}^t(\alpha_{\mathscr{I}}) \quad \text{where} \quad D_{\mathscr{I}}^t(\alpha_{\mathscr{I}}) = \frac{1}{n}\sum_{i\in\mathscr{I}}\Big(y_i + y_i\log\tfrac{\alpha_i}{y_i}\Big) - \frac{\lambda}{2}\Big\| w^t + \frac{1}{\lambda n}\sum_{i\in\mathscr{I}}(\alpha_i-\alpha_i^t)x_i\Big\|^2,$$

where we denote by $\alpha_{\mathscr{I}}$ the sub-vector of α of size p containing the values of all indices in $\mathscr{I}$. We initialize the vector $\alpha_{\mathscr{I}}^{(0)}\in(-\mathscr{D}_{f^*})^p$ to the corresponding values of the coordinates of α^t in $\mathscr{I}$ and then perform the Newton steps, i.e.

$$\alpha_{\mathscr{I}}^{k+1} = \alpha_{\mathscr{I}}^k - \Delta\alpha_{\mathscr{I}}^k \quad \text{where } \Delta\alpha_I^k \text{ is the solution of} \quad \nabla^2 D_{\mathscr{I}}^t(\alpha_{\mathscr{I}}^k)\Delta\alpha_{\mathscr{I}}^k = \nabla D_{\mathscr{I}}^t(\alpha_{\mathscr{I}}^k). \tag{IV.13}$$

The gradient $\nabla D_{\mathscr{I}}^t(\alpha_{\mathscr{I}}^k)$ and the hessian $\nabla^2 D_{\mathscr{I}}^t(\alpha_{\mathscr{I}}^k)$ have the following explicit formulas:

$$(\nabla D_{\mathscr{I}}^t(\alpha_{\mathscr{I}}^k))_i = \frac{\partial D(\alpha_{\mathscr{I}}^k)}{\partial\alpha_i} = \frac{1}{n}\Big(\frac{y_i}{\alpha_i^k} - w^{t\top}x_i - \frac{1}{\lambda n}\sum_{j\in\mathscr{I}}(\alpha_j^k-\alpha_j^t)x_j^\top x_i\Big),$$

and

$$\big(\nabla^2 D_I^t(\alpha_I^k)\big)_{i,j} = \frac{\partial^2 D(\alpha_I^k)}{\partial\alpha_i\partial\alpha_j} = -\frac{1}{n}\Big(\frac{y_i}{{\alpha_i^k}^2}\mathbf{1}_{i=j} + \frac{1}{\lambda n}x_i^\top x_j\Big).$$

Note that D_I^t is a concave function hence $-\nabla^2 D_I^t(\alpha_I^k)$ will be positive semi-definite and the system in Equation (IV.13) can be solved very efficiently with BLAS and LAPACK libraries. Let us explicit computations when $p=2$. Suppose that $\mathscr{I}=\{i,j\}$ and put

$$\delta_i = \alpha_i - \alpha_i^{(t-1)}, \quad p_i = x_i^\top w^{(t-1)} \quad \text{and} \quad g_{ij} = \frac{x_i^\top x_j}{\lambda n}.$$

The gradient and the Hessian inverse are then given by

$$\nabla D(\alpha_I) = \frac{1}{n}\begin{bmatrix} \frac{y_i}{\alpha_i} - p_i - \delta_i g_{ii} - \delta_j g_{ij} \\ \frac{y_j}{\alpha_j} - p_j - \delta_j g_{jj} - \delta_i g_{ij}\end{bmatrix},$$

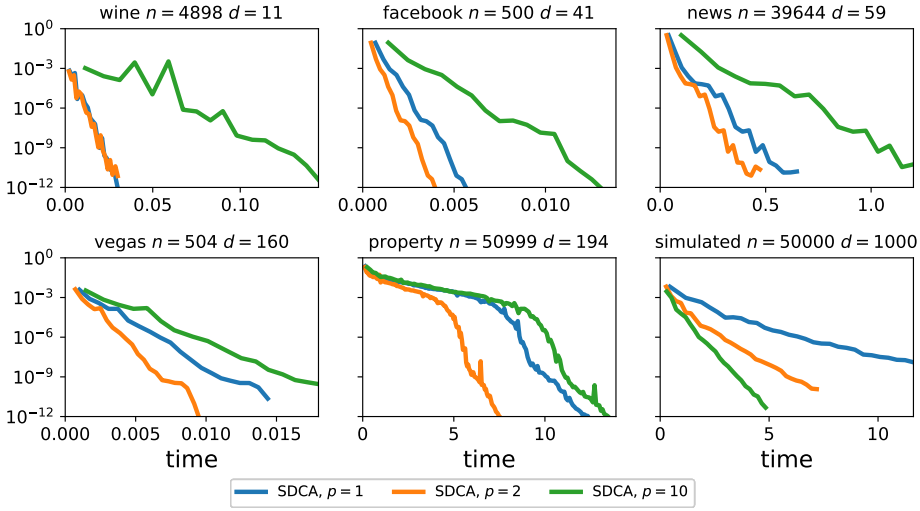

Figure IV.10: Convergence speed comparison when the number of indices optimized at each step changes.

and

$$\nabla^2 D(\alpha_I)^{-1} = \frac{n^2}{(\frac{y_i}{\alpha_i^2} + g_{ii})(\frac{y_i}{\alpha_i^2} + g_{ii}) - g_{ij}^2} \begin{bmatrix} -\frac{y_j}{\alpha_j^2} - g_{jj} & g_{ij} \\ g_{ij} & -\frac{y_i}{\alpha_i^2} - g_{ii} \end{bmatrix}.$$

This direct computation leads to even faster computations than using the dedicated libraries. We plot in Figure IV.10 the convergence speed for three sizes of batches 1, 2 and 10. Note that in all cases using a batch of size $p = 2$ is faster than standard SDCA. Also, in the last simulated experiment where d has been set on purpose to 1000, the solver using batches of size $p = 10$ is the fastest one. The bigger number of features d gets, the better are solvers using big batches.

5.5 About the pessimistic upper bounds

The generic upper bounds derived in Proposition 5 are general but pessimistic as they depend linearly on n. In fact this dependence is also observed in the NoLips algorithm [BB16] where the rate depends on a constant $L = \sum_i^n y_i$. Note that, for Nolips algorithms, L is involved in the step size definition and leads to step too small to be used in practice but that in our algorithm, these bounds have little or even no impact in practice (see Remark 2) and are mainly necessary for convergence guarantees. These bounds are derived by lower bounding $x_i^\top \sum_{j \neq i} \alpha_j^* x_j$ by 0. This lower bound is very conservative and can probably be tightened by setting specific hypotheses on the dataset, for example on the Gram matrix ($[G]_{i,j} = x_i^\top x_j$ for $i, j = 1, \ldots, n$). For Poisson

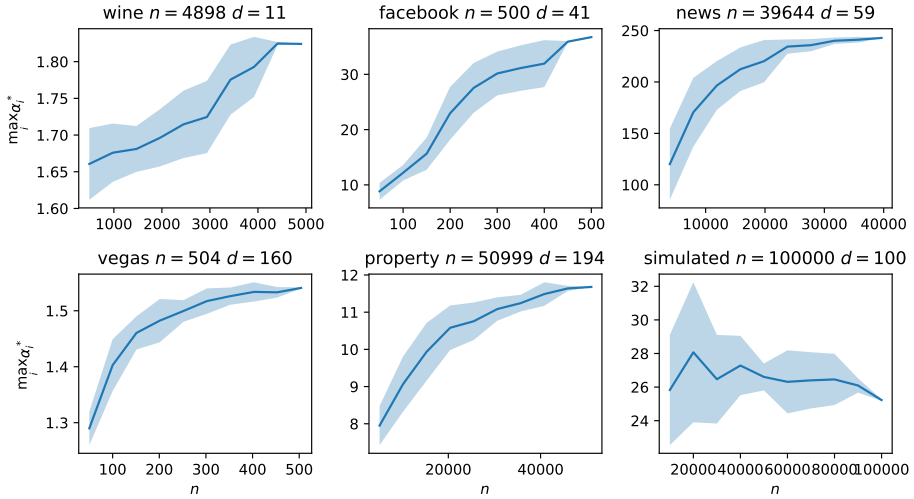

Figure IV.11: Evolution of $\max_{i=1,\dots,n} \alpha_i^*$ for an increasing value of n. We observe that $\max_{i=1,\dots,n} \alpha_i^*$ is not increasing linearly with n as quick as the bound β_i obtained in Proposition 5.

regression, this lower bound is reached in the extreme case where all observations are orthogonal (all entries of G are zero except on the diagonal). Then $\psi^\top x_i = \frac{1}{n}\|x_i\|^2$ and the upper bounds from Proposition 5 become

$$\beta_i = \frac{1}{2} + \sqrt{\frac{1}{4} + \frac{\lambda n y_i}{\|x_i\|^2}}$$

for $i = 1, \dots, n$. In this extreme case, the bounds are $\mathcal{O}(\sqrt{n})$ instead of $mathcalO(n)$ as stated in Proposition 5. Experimentally, we do not observe a dependence in $\mathcal{O}(n)$ either. Figure IV.11 shows the evolution of the maximum optimal dual obtained ($\max_{i=1,\dots,n} \alpha_i^*$) for the six datasets considered in Section 5.1 for Poisson regression, on an increasing fraction of the dataset. These values are averaged over 20 samples and we provide the associated 95% confidence interval on this value. We observe that $\max_{i=1,\dots,n} \alpha_i^*$ has a much lower dependence in n than the bounds given by Proposition 5.

6 Conclusion

This work introduces the log-smoothness assumption in order to derive improved linear rates for SDCA, for objectives that do not meet the gradient-Lipschitz assumption. This provides, to the best of our knowledge, the first linear rates for a stochastic

first order algorithm without the gradient-Lipschitz assumption. The experimental results also prove the efficiency of SDCA to solve such problems and its ability to deal with the open polytope constraints, improving the state-of-the-art. Finally, this work also presents several variants of SDCA and experimental heuristics to make the most of it on real world datasets. Future work could provide better linear rates under more specialized assumptions on the Gram matrix, as observed on numerical experiments. Also, to extend this work to more applications, we aim to find a generalization of the log smoothness assumption such as what [STD17] has done for self-concordance.

Acknowledgments

We would like to acknowledge support for this project from the Datascience Initiative of École polytechnique

7 Proofs

We start this Section by providing extra details on the derivation of the dual problem and the proximal version of SDCA. We then provides the proofs of all the results stated in the chapter, namely Proposition 2, Proposition 3, Theorem 1, Theorem 2, Proposition 4, Proposition 5, Remark 2, Proposition 6 and Proposition 7.

7.1 Duality and proof of Proposition 1

It is not straightforward to obtain strong duality for a convex problem with strict inequalities (such as the ones enforced by the polytope $\Pi(X)$ from Equation IV.3). To bypass this difficulty we consider the same problem but constrained on a closed set and show how it relates to the original Problem (IV.2). But first we formulate the two following lemmas.

Lemma 1. *There exists $\varepsilon > 0$ such that the following problem*

$$\min_{w \in \Pi_{|\varepsilon}(X)} P(w) \quad \textit{where} \quad \Pi_{|\varepsilon}(X) = \{w \in \mathbb{R}^d \,:\, \forall i \in \{1,\dots,n\},\ w^\top x_i \geq \varepsilon\}$$

has a solution w^* that is also the solution of the original Problem (IV.2).

Proof. First, notice that this problem has a unique solution which is unique as we minimize a strongly convex function on a closed convex set $\Pi_{|\varepsilon}(X)$. Also, as the function $w \mapsto \psi^\top w + \lambda g(w)$ is strongly convex, since g is strongly convex, it is lower bounded. We denote by $M \in \mathbb{R}$ a lower bound of this function. Then, we consider $w^0 \in \Pi(X)$, and choose ε sufficiently small such that for all $i = 1,\dots,n$,

$$\forall t < \varepsilon,\ f_i(t) > nP(w^0) - nM,$$

this value of ε always exists since by assumption $\lim_{t\to 0} f_i(t) = +\infty$ for all $i = 1,\dots,n$. For any $w_\varepsilon \in \Pi(X)\setminus\Pi_\varepsilon(X)$ (so a w_ε is such that $\exists i \in \{1,\dots,n\},\ w_\varepsilon^\top x_i < \varepsilon$), we thus have

$$P(w_\varepsilon) > \psi^\top w_\varepsilon + P(w^0) - M + \lambda g(w_\varepsilon) \geq P(w^0).$$

Hence, for such a value of ε, the solution to the original Problem (IV.2) is necessarily in $\Pi_\varepsilon(X)$ and both the problems constrained on $\Pi_\varepsilon(X)$ and $\Pi(X)$ share the same solution w^*. ■

Lemma 2. *For all $i = 1,\dots,n$, if we define by*

$$\forall \alpha_i \in \mathscr{D}_{f^*},\ f^*_{i|\varepsilon}(v) := \max_{u\geq\varepsilon} uv - f_i(u),$$

then $f^*_{i|\varepsilon}$ is equal to the Fenchel conjugate of f_i, f_i^*, on $\{v\,;\,\exists u \geq \varepsilon\,;\ v = f_i'(u)\}$.

Proof. For all $i = 1,\dots,n$, the functions f_i are convex and differentiable. Hence, by Fermat's rule if $\exists u^* \geq \varepsilon;\ v = f_i'(u^*)$ then $f^*_{i|\varepsilon}(v) = \max_{u\geq\varepsilon} uv - f_i(u) = u^* f_i'(u^*) - f_i(u^*)$. Likewise, the maximization step in the computation of f_i^* would share the same maximizer and $f_i^*(v) = u^* f_i'(u^*) - f_i(u^*)$ as well. Hence,

$$\forall v \text{ such that } \exists u \geq \varepsilon\,;\ v = f_i'(u);\ f^*_{i|\varepsilon}(v) = f_i^*(v).$$

■

We form the dual of the problem constrained on $\Pi_\varepsilon(X)$ where ε is such that Lemma 1 applies. We replace the inner products $w^\top x_i$ by the scalars u_i for $i = 1,\dots,n$ and their equality is constrained to form the strictly equivalent problem:

$$\min_{\substack{w\in\mathbb{R}^d, u\in[\varepsilon,+\infty)^n \\ \forall i, u_i = w^\top x_i}} \psi^\top w + \frac{1}{n}\sum_{i=1}^n f_i(u_i) + \lambda g(w).$$

We maximize the Lagrangian to include the constraints. This introduces the vector of dual variables $\alpha \in \mathbb{R}^n$ as following:

$$\max_{\alpha\in\mathbb{R}^n} \min_{\substack{w\in\mathbb{R}^d \\ u\in[\varepsilon,+\infty)^n}} \psi^\top w + \frac{1}{n}\sum_{i=1}^n f_i(u_i) + \lambda g(w) + \frac{1}{n}\sum_{i=1}^n \alpha_i(u_i - w^\top x_i)$$

that leads to the corresponding dual problem:

$$\max_{\alpha\in(-\mathscr{D}_{f^*_{|\varepsilon}})^n} D_{|\varepsilon}(\alpha), \quad D_{|\varepsilon}(\alpha) = \frac{1}{n}\sum_{i=1}^n -f^*_{i|\varepsilon}(-\alpha_i) - \lambda g^*\Big(\frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i - \frac{1}{\lambda}\psi\Big),$$

where $-\mathscr{D}_{f^*_{|\varepsilon}}$ is the domain of all $f^*_{i|\varepsilon}$. The primal problem constrained on $\Pi_\varepsilon(X)$ verifies the Slater's conditions so strong duality holds and the maximizer of $D_{|\varepsilon}$,

$\alpha^*_{|\varepsilon}$, is reached. As $D_{|\varepsilon}$ is concave, $\alpha^*_{|\varepsilon}$ is the only vector such that $\nabla D_{|\varepsilon}(\alpha^*_{|\varepsilon}) = 0$. Also, as strong duality holds, we can relate $\alpha^*_{|\varepsilon}$ to the primal optimum though the Karush-Kuhn-Tucker condition

$$\alpha^*_{i|\varepsilon} = -f^{*\prime}_{i|\varepsilon}(w^{*\top} x_i),$$

where w^* is such that $w^{*\top} x_i \geq \varepsilon$ (see Lemma 1). Hence Lemma 2 applies and the dual formulation of the original Problem (IV.2) that writes

$$\max_{\alpha \in (-\mathscr{D}_{f^*})^n} D(\alpha), \quad D(\alpha) = \frac{1}{n}\sum_{i=1}^n -f_i^*(-\alpha_i) - \lambda g^*\Big(\frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i - \frac{1}{\lambda}\psi\Big),$$

is such that $\nabla D(\alpha^*_{|\varepsilon}) = \nabla D_{|\varepsilon}(\alpha^*_{|\varepsilon}) = 0$. Since $D(\alpha)$ is concave, this means that $\alpha^*_{|\varepsilon} = \alpha^*$ where α^* is the solution of the dual formulation of the original Problem (IV.2). Thus, the Karush-Kuhn-Tucker conditions that link the primal and dual optima of the problem constrained on $\Pi_{|\varepsilon}(X)$ also links the primal and dual optima of the original Problem (IV.2). The first one is given in Equation (IV.5) and the second one writes

$$\alpha^*_i = -f^{*\prime}_i(w^{*\top} x_i) \tag{IV.14}$$

for any $i \in \{1, \dots n\}$. From the first we can define two functions linking vector $w \in \mathbb{R}^d$ to $\alpha \in (-\mathscr{D}_{f^*})^n$ and such that $w(\alpha^*) = w^*$ and

$$v(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i - \frac{1}{\lambda}\psi \quad \text{and} \quad w(\alpha) = \nabla g^*\big(v(\alpha)\big). \tag{IV.15}$$

7.2 Proximal algorithm

Given that g^* is smooth since its Fenchel conjugate is strongly convex, the gradient-Lipschitz property from Definition 4 below entails $g^*(v + \Delta v) \leq g^*(v) + \nabla g^*(v)^\top \Delta v + \frac{1}{2}\|\Delta v\|^2$. Hence, maximization step of Algorithm IV.1, namely,

$$\operatorname*{arg\,max}_{\alpha_i \in -\mathscr{D}_{f^*}} -f_i^*(-\alpha_i) - \lambda n g^*\big(v^{(t-1)} + (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i\big),$$

where $v^{(t-1)} = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^{(t-1)} x_i - \frac{1}{\lambda}\psi$ can be simplified by setting α_i^t such that it maximizes the lower bound

$$\alpha_i^t = \operatorname*{arg\,max}_{\alpha_i \in -\mathscr{D}_{f^*}} -f_i^*(-\alpha_i) - \lambda n\Big(g^*\big(v^{(t-1)}\big) + \frac{\alpha_i - \alpha_i^{(t-1)}}{\lambda n} x_i^\top \nabla g^*(v^{(t-1)}) + \frac{1}{2}\Big(\frac{\alpha_i - \alpha_i^{(t-1)}}{\lambda n}\Big)^2 \|x_i\|^2\Big). \tag{IV.16}$$

Setting $w^{(t-1)} = \nabla g^*(v^{(t-1)})$ and discarding constants terms leads to the equivalent relation,

$$\alpha_i^t = \underset{\alpha_i \in -\mathscr{D}_{f^*}}{\arg\max} -f_i^*(-\alpha_i) - \frac{\lambda n}{2}\left\| w^{(t-1)} - (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i \right\|^2.$$

While convergence speed is guaranteed for any 1-strongly convex g, to simplify the algorithm we will consider that g is not only 1-strongly convex but that that it can also be decomposed as

$$g(w) = \tfrac{1}{2}\|w\|^2 + h(w)$$

where h is a prox capable function. With Proposition 8 below the relation between w^t and v^t becomes

$$w^t = \nabla g^*(v^t) = \underset{u \in \mathbb{R}^d}{\arg\sup}\left(u^\top v^t - \tfrac{1}{2}\|u\|^2 - h(u)\right) = \underset{u \in \mathbb{R}^d}{\arg\inf}\left(\tfrac{1}{2}\|v^t - u\|^2 + h(u)\right),$$

which is the proximal operator stated in Definition 6 below: $w^t = \mathrm{prox}_h(v(\alpha^t))$.

7.3 Preliminaries for the proofs

Let us first recall some definitions and basic properties.

Definition 3. Strong convexity. *A differentiable convex function $f : \mathscr{D}_f \to \mathbb{R}$ is γ-strongly convex if*

$$\forall x, y \in \mathscr{D}_f, \quad f(y) \geq f(x) + f'(x)^\top (y - x) + \frac{\gamma}{2}\|y - x\|^2. \tag{IV.17}$$

This is equivalent to

$$\forall x, y \in \mathscr{D}_f, \quad (f'(y) - f'(x))(y - x) \geq \gamma \|y - x\|^2. \tag{IV.18}$$

Definition 4. Smoothness. *A differentiable convex function $f : \mathscr{D}_f \to \mathbb{R}$ is L-smooth or L-gradient-Lipschitz if*

$$\forall x, y \in \mathscr{D}_f, \quad f(y) \leq f(x) + f'(x)(y - x) + \frac{L}{2}\|y - x\|^2.$$

This is equivalent to

$$\forall x, y \in \mathscr{D}_f, \quad (f'(y) - f'(x))(y - x) \leq L\|y - x\|^2.$$

Definition 5. Fenchel conjugate. *For a convex function $f : \mathscr{D}_f \to \mathbb{R}$ we call Fenchel conjugate the function f^* defined by*

$$f^* : \mathscr{D}_{f^*} \to \mathbb{R}, \quad \textit{st.} \quad f^*(v) = \sup_{u \in \mathscr{D}_f} \left(u^\top v - f(u)\right). \tag{IV.19}$$

Proposition 8. *For a convex differentiable function f, the gradient of its differentiable Fenchel conjugate f^* is the maximizing argument of* (IV.19):

$$f^{*\prime}(v) = \operatorname*{arg\,sup}_{u \in \mathscr{D}_f} \big(u^\top v - f(u) \big).$$

Proposition 9. *For a convex differentiable function f and its differentiable Fenchel conjugate f^* we have*

$$\forall u \in \mathscr{D}_f, f^{*\prime}(f'(u)) = u \quad \textit{and} \quad \forall v \in \mathscr{D}_{f^*}, f'(f^{*\prime}(v)) = v.$$

This leads to

$$\forall u \in \mathscr{D}_f, \forall v \in \mathscr{D}_{f^*}, \quad f'(u) = v \Leftrightarrow u = f^{*\prime}(v).$$

Note that if f is γ-strongly convex (respectively L-smooth), then its Fenchel conjugate f^* is $1/\gamma$ smooth (respectively $1/L$ strongly convex). We also recall results from [NN94] on self-concordant functions introduced in Definition 2. This concept is widely used to study losses involving logarithms. For the sake of clarity, the results will be presented for functions whose domain $\mathscr{D}_f$ is a subset of $\mathbb{R}$ as this leads to lighter notations. Unlike smoothness and strong convexity, this property is affine invariant. From this definition, some inequalities are derived in [NN94]. Two of them provide lower bounds that are comparable to strong convexity inequalities:

$$\forall x, y \in \mathscr{D}_f, \quad f(y) \geq f(x) + f'(x)(y-x) + \omega\big(\sqrt{f''(x)}|y-x|\big) \tag{IV.20}$$

where $\omega(t) = t - \log(1+t)$, and

$$\forall x, y \in \mathscr{D}_f, \quad (f'(y) - f'(x))^\top (y-x) \geq \frac{f''(x)(y-x)^2}{1 + \sqrt{f''(x)}|y-x|}. \tag{IV.21}$$

Finally, we define the proximal operator used to apply the penalization g.

Definition 6. Proximal operator. *For a convex function $g : \mathscr{D}_g \to \mathbb{R}$, where $\mathscr{D}_g$ is closed, the proximal operator associated to g is given by*

$$\operatorname{prox}_g(y) = \operatorname*{arg\,min}_{x \in \mathscr{D}_g} \Big(\frac{1}{2}\|y-x\|^2 + g(x) \Big).$$

The proximal operator always exists and is uniquely defined as the minimizer of a strongly convex function. Before the proof of Theorem 1, we need to introduce new convex inequalities for L-log smooth functions. This class of function includes $x \mapsto -L\log x$ which is our function of interest in the Poisson and in the Hawkes cases.

7.4 Proof of Proposition 2

First order implies second order We start by showing that if f is a L log-smooth function then we can bound its second derivative by the square of its gradient. For any $x \in \mathscr{D}_f$, we set $y = x + h$ in the Definition 1, which now writes

$$\forall x \in \mathscr{D}_f,\ \forall h \text{ s.t. } (x+h) \in \mathscr{D}_f,\ \left|\frac{f'(x) - f'(x+h)}{h}\right| \leq \frac{1}{L} f'(x) f'(x+h).$$

Taking the limit of the previous inequality when h tends to 0 leads to the desired inequality,

$$\forall x \in \mathscr{D}_f,\ \left|f''(x)\right| \leq \frac{1}{L} f'(x)^2.$$

Second order implies first order We now prove that if f is convex strictly monotone, twice differentiable and $|f''(x)| \leq \frac{1}{L} f'(x)^2$ then f is L-log smooth. If for all $x \in \mathscr{D}_f$, we denote by $\phi : x \mapsto \frac{1}{f'(x)}$, (note that $\forall x \in \mathscr{D}_f, f'(x) \neq 0$ as f is strictly monotone), then

$$\forall x \in \mathscr{D}_f,\ |\phi'(x)| = \left|\frac{f''(x)}{f'(x)^2}\right| \leq \frac{1}{L}.$$

From this inequality, we limit the increasings of the function ϕ,

$$\forall x, y \in \mathscr{D}_f,\ -\tfrac{1}{L}|y - x| \leq \phi(y) - \phi(x) \leq \tfrac{1}{L}|y - x|,$$

which rewrites

$$\forall x, y \in \mathscr{D}_f,\ |\phi(y) - \phi(x)| \leq \tfrac{1}{L}|y - x| \ \Leftrightarrow\ \left|\frac{f'(x) - f'(y)}{f'(x) f'(y)}\right| \leq \frac{1}{L}|x - y|,$$

that is the definition of a L-log smooth function for a convex strictly monotone function.

7.5 Proof of Proposition 3

We working by exhibiting several statements equivalent to log smoothness. First, we divide both sides of the log smoothness definition by $f'(x) f'(y) > 0$ since f is strictly monotone,

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_f,\ \left|\frac{1}{f'(y)} - \frac{1}{f'(x)}\right| \leq \frac{1}{L}|x - y|.$$

Then we rewrite the equation in the dual space using Proposition 9,

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_{f^*},\ \left|\frac{1}{y} - \frac{1}{x}\right| \leq \frac{1}{L}|f^{*\prime}(x) - f^{*\prime}(y)|.$$

This can be rewritten into the following integrated form

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_{f^*},\ \left|\int_y^x t^{-2} dt\right| \leq \frac{1}{L}\left|\int_y^x (f^{*\prime\prime}(t)\,\mathrm{d}t\right|,$$

which becomes equivalent to the desired result with the fundamental theorem of calculus

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x \in \mathscr{D}_{f^*},\ x^{-2} \leq \frac{1}{L} f^{*\prime\prime}(x).$$

7.6 Inequalities for log-smooth functions

The proof of SDCA [SSZ13] relies on the smoothness of the functions f_i which implies strong convexity of their Fenchel conjugates f_i^*. Indeed, a γ strongly convex function f^* satisfies the following inequality

$$s f^*(x) + (1-s) f^*(y) \geq f^*(sx + (1-s)y) + \frac{\gamma}{2} s(1-s)(y-x)^2. \tag{IV.22}$$

This inequality is not satisfied for L-log smooth functions. However, we can derive for such functions another inequality which can be compared to such inequalities based on self-concordance and strongly convex properties.

Lemma 3. *Let $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ be a strictly monotone convex function and f^* be its differentiable Fenchel conjugate. Then,*

$$f \textit{ is } L\text{-log } \textit{smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_{f^*},\ (f^{*\prime}(x) - f^{*\prime}(y))(x-y) \geq L\frac{(x-y)^2}{xy}.$$

This bound is an equality for $f(x) = -L\log x$.

Proof. From log smoothness definition, we obtain by multiplying both sides by $|f'(x) - f'(y)|$ and dividing by $f'(x)f'(y) > 0$ (since f is strictly monotone),

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_f,\ \frac{\big(f'(x) - f'(y)\big)^2}{f'(x)f'(y)} \leq \frac{1}{L}|x-y||f'(x) - f'(y)|$$

Since f is a convex function, $(f'(x) - f'(y))(x-y) \geq 0$ and using Proposition 9, we can rewrite the previous equivalence in the dual space:

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_{f^*},\ \frac{(x-y)^2}{xy} \leq \frac{1}{L}(f^{*\prime}(x) - f^{*\prime}(y))(x-y),$$

which concludes the proof. ∎

Lemma 4. *Let $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ be a strictly monotone convex function and f^* be its differentiable Fenchel conjugate. Then,*

$$f \text{ is } L\text{-log } \textit{smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_{f^*},\ f^*(x) - f^*(y) - f^{*\prime}(y)(x-y) \geq L\Big(\frac{x}{y} - 1 - \log\frac{x}{y}\Big).$$

This bound is an equality for $f(x) = -L\log x$.

Proof. Let $x, y \in \mathscr{D}_{f^*}$, we have the following on the one hand,

$$f^*(x) - f^*(y) - f^{*\prime}(y)(x-y) = \int_y^x \big(f^{*\prime}(u) - f^{*\prime}(y)\big)\,\mathrm{d}u$$

On the other hand, applying Lemma 3 together with the fundamental theorem of calculus gives

$$f \text{ is } L\text{-log smooth} \quad \Leftrightarrow \quad \forall x, y \in \mathscr{D}_{f^*},\ \int_y^x (f^{*\prime}(u) - f^{*\prime}(y))\,\mathrm{d}u \geq \int_y^x L\,\frac{u-y}{uy}\,\mathrm{d}u.$$

Finally, solving the integral leads to the desired result:

$$\forall x, y \in \mathscr{D}_{f^*},\ \int_y^x L\,\frac{u-y}{uy}\,\mathrm{d}u = L\int_y^x \Big(\frac{1}{y} - \frac{1}{u}\Big)\,\mathrm{d}u = L\Big(\frac{x-y}{y} - \log\frac{x}{y}\Big).$$

■

Lemma 5. *Assume that f is L-log smooth and f^* is its differentiable Fenchel conjugate, then,*

$$sf^*(x) + (1-s)f^*(y) - f^*(y + s(x-y)) \geq L\Big(\log\Big(1 - s + s\frac{x}{y}\Big) - s\log\frac{x}{y}\Big)$$

for any $y, x \in \mathscr{D}_{f^*}$ and $s \in [0,1]$. This bound is an equality for $f(x) = -L\log x$.

Proof. Let $x, y \in \mathscr{D}_{f^*}$ and define for any $s \in [0,1]$, $u(s) = sx + (1-s)y$. We apply Lemma 4 twice for x, $u(s)$ and y, $u(s)$:

$$f^*(x) - f^*(u(s)) - f^{*\prime}(u(s))(x - u(s)) \geq L\Big(\frac{x}{u(s)} - 1 - \log\frac{x}{u(s)}\Big), \tag{IV.23}$$

$$f^*(y) - f^*(u(s)) - f^{*\prime}(u(s))(y - u(s)) \geq L\Big(\frac{y}{u(s)} - 1 - \log\frac{y}{u(s)}\Big). \tag{IV.24}$$

Combining s(IV.23) and $(1-s)$(IV.24) leads to

$$\begin{aligned} sf^*(x)+(1-s)f^*(y)-f^*(u(s)) &\geq -sL\log\frac{x}{u(s)}-(1-s)L\log\frac{y}{u(s)} \\ &\quad + L\Big(s\frac{x}{u(s)}+(1-s)\frac{y}{u(s)}-1\Big) \\ &= sL\log\frac{u(s)}{x}+(1-s)L\log\frac{u(s)}{y} \\ &= sL\log\frac{u(s)}{y}+sL\log\frac{y}{x}+(1-s)L\log\frac{u(s)}{y} \\ &= L\log\Big(1-s+s\frac{x}{y}\Big)+sL\log\frac{y}{x}. \end{aligned}$$

■

This Lemma which implies the barycenter $u(s)=y+s(x-y)$ for $s\in[0,1]$ is the lower bound that we actually use in proof of Theorem 1. To compare this result with strong convexity and self-concordance assumptions, we will suppose that f^* is twice differentiable and hence that Proposition 3 applies.

Comparison with self-concordant functions Instead of building our lower bounds on log smoothness, we rather exhibit what can be obtained with self-concordance combined with the lower bound $f^{*\prime\prime}(y)\geq Ly^{-2}$ from Proposition 3. In this paragraph, we consider that $\frac{1}{L}f^*$ is standard self-concordant, such an hypothesis is verified for $f:t\mapsto-\frac{1}{L}\log(t)$. Hence, using lower bound (IV.21) on $\frac{1}{L}f$ and then Proposition 3, we obtain

$$\forall x,y\in\mathscr{D}_{f^*},\ (f^{*\prime}(x)-f^{*\prime}(y))(x-y)\geq\frac{f^{*\prime\prime}(y)(x-y)^2}{1+\sqrt{\frac{1}{L}f^{*\prime\prime}(y)}|x-y|}\geq L\frac{(x-y)^2}{y^2+|y(x-y)|}.$$

Since $\forall x,y\in\mathscr{D}_{f^*},\ xy>0$, this lower bound is equivalent to Lemma 3 if $|x|\geq|y|$ but not as good otherwise. Lemma 4 can also be compared to what can be obtained applying Inequality (IV.20) on $\frac{1}{L}f$. Since $\omega:t\mapsto t-\log(1+t)$ is an increasing function, it leads to

$$\forall x,y\in\mathscr{D}_{f^*},\ f^*(x)-f^*(y)-f^{*\prime}(y)(x-y)\geq L\,\omega\Big(\sqrt{\tfrac{1}{L}f^{*\prime\prime}(y)}\,|x-y|\Big)\geq L\ \omega\Big(\Big|\tfrac{x}{y}-1\Big|\Big).$$

Again, this lower bound the same as Lemma 4 if $|x|\geq|y|$ but not as good otherwise. Finally, a bound equivalent to Lemma 5 for self-concordant functions is not easy to explicit in a clear form. However, it is numerically smaller than the lower bound stated in Lemma 5 for any $s\in[0,1]$ and any $x,y\in\mathscr{D}_{f^*}$.

	strongly convex	self-concordant	log smoothness
Lemma 3	$\frac{(x-y)^2}{\max(x^2,y^2)}$	$\frac{(x-y)^2}{y^2+\lvert y(x-y)\rvert}$	$\frac{(x-y)^2}{xy}$
Lemma 4	$\frac{(x-y)^2}{2\max(x^2,y^2)}$	$\lvert\frac{x}{y}-1\rvert-\log\big(1+\lvert\frac{x}{y}-1\rvert\big)$	$\frac{x}{y}-1-\log\big(\frac{x}{y}\big)$
Lemma 5	$s(1-s)\frac{(x-y)^2}{2\max(x^2,y^2)}$	–	$\log\big(1-s+s\frac{x}{y}\big)+s\log\frac{y}{x}$
Reached for $f=-\log$	✗	✗	✓

Table IV.3: Comparison of lower bounds obtained with different hypotheses. These lower bounds come from Lemmas 3, 4 and 5. It shows that both the strongly-convex and self-concordant hypotheses are not enough to reach the inequality obtained under log smoothness. The inequality coming from Lemma 5 is missing as it cannot be easily exhibited for self-concordant functions.

Comparison with strongly convex functions We cannot directly assume that f^* is strongly convex as it would mean that f is gradient-Lipschitz. But, for fixed values of x and $y \in \mathscr{D}_{f^*}$, we define on $\{u \in \mathscr{D}_{f^*}, |u| < \max(|x|,|y|)\}$ the function $f^*_{\{x,y\}} : u \mapsto f^*(u)$ as the restriction of f^* on this interval. The lower bound $f^{*\prime\prime}(y) \geq Ly^{-2}$ from Proposition 3 implies that $f^*_{\{x,y\}}$ is $L/\max(x^2,y^2)$ strongly-convex on its domain to which x and y belong. Equation (IV.18) leads to the following inequality, valid for $f^*_{\{x,y\}}$ and thus for f^*,

$$\forall x, y \in \mathscr{D}_{f^*},\ (f^{*\prime}(x) - f^{*\prime}(y))(x-y) \geq L\frac{(x-y)^2}{\max(x^2,y^2)}.$$

As soon as $x \neq y$, this lower bound is not as good as the one provided by Lemma 3. Following the same logic, we exhibit the two following lower bounds. The first one corresponds to Lemma 4 and is entailed by Equation (IV.17),

$$\forall x, y \in \mathscr{D}_{f^*},\ f^*(x) - f^*(y) - f^{*\prime}(y)(x-y) \geq \frac{L}{2}\frac{(x-y)^2}{\max(x^2,y^2)},$$

and the second to Lemma 5 and is entailed by Equation (IV.22)

$$\forall x, y \in \mathscr{D}_{f^*},\ \forall s \in [0,1],\ sf^*(x) + (1-s)f^*(y) - f^*(y + s(x-y)) \geq s(1-s)\frac{L}{2}\frac{(x-y)^2}{\max(x^2,y^2)}.$$

In both cases the reached lower bounds are not as tight as the ones stood in Lemmas 4 and 5. All these bounds are reported in Table IV.3 for an easy comparison.

Finally, two lemmas to lower bound Lemma 5 are needed as well.

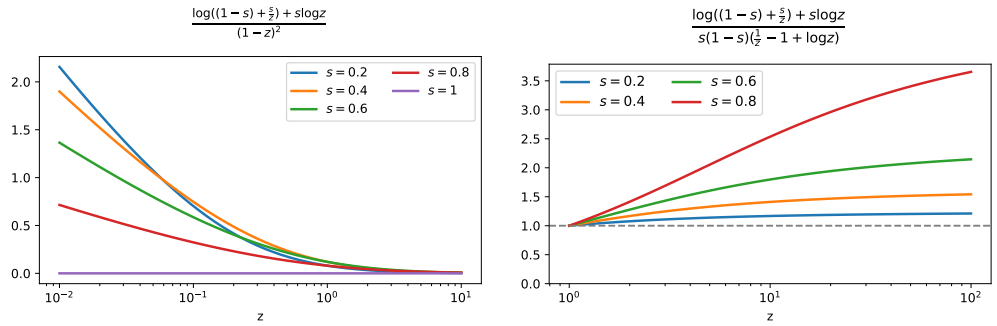

Figure IV.12: Illustrations of Lemma 6 and Lemma 7.
Left: Lemma 6, $\frac{\log\left((1-s)+\frac{s}{z}\right)+s\log z}{(1-z)^2}$ is decreasing in z for any fixed $s\in[0,1]$.
Right: Lemma 7, $\log\left((1-s)+\frac{s}{z}\right)+s\log z \geq s(1-s)\left(\frac{1}{z}-1+\log z\right)$ for any $z\geq 1$ and $s\in[0,1]$.

Lemma 6. *The function f defined by*

$$f(s,z)=\frac{\log\left((1-s)+\frac{s}{z}\right)+s\log z}{(1-z)^2}$$

for all $z\in\mathbb{R}^{++}$ and $s\in[0,1]$ is a decreasing function in z.

Lemma 7. *We have*

$$\log\left((1-s)+\frac{s}{z}\right)+s\log z\geq s(1-s)\left(\frac{1}{z}-1+\log z\right)$$

for all $z\geq 1$ and $s\in[0,1]$.

The analytical proof of these lemmas are very technical and not much informative. Thus, we rather illustrate them with the two following figures

7.7 Proof of Theorem 1

This proof is very similar to SDCA's proof [SSZ14] but it uses the new convex inequality on the Fenchel conjugate of log smooth functions from Lemma 5 to get a tighter inequality. We first prove the following lemma which is an equivalent of Lemma 6 from [SSZ14] but with convex functions f_i that are L_i-log smooth instead of being L_i-gradient-Lipschitz.

Lemma 8. *Suppose that we known bounds $\beta_i\in-\mathscr{D}_{f^*}$ such that $R_i=\beta_i/\alpha_i^*\geq 1$ for $i=1,\dots,n$ and assume that all f_i are L_i-log smooth with differentiable Fenchel conjugates*

and that g is 1-strongly convex. Then, if $\alpha^{(t,i)}$ is the value of $\alpha^{(t)}$ when i is sampled at iteration t for Algorithms IV.1 and IV.2, we have

$$\sum_{i=1}^{n} s_i^{-1}\big(D(\alpha^{(t,i)}) - D(\alpha^{(t-1)})\big) \geq D(\alpha^*) - D(\alpha^{(t-1)}) + G(s_i, \alpha_i^{(t-1)}, \alpha_i^*) \tag{IV.25}$$

for any $s_1, \dots, s_n \in [0,1]$, where

$$G(s, \alpha^{(t-1)}, \alpha^*) = \frac{1}{n}\sum_{i=1}^{n}\left(L_i\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) - \frac{s_i}{2\lambda n}\|x_i\|^2(\alpha_i^* - \alpha_i^{(t-1)})^2\right)$$

and

$$\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) = \frac{1}{s_i}\log\Big(1 - s_i + s_i\frac{\alpha_i^*}{\alpha_i^{(t-1)}}\Big) - \log\frac{\alpha_i^*}{\alpha_i^{(t-1)}}.$$

Proof. At iteration t, if the dual vector is set to $\alpha^{(t)}$ (and $v^{(t)} = v(\alpha^{(t)})$, see Equation (IV.15)) the dual gain is

$$n(D(\alpha^{(t)}) - D(\alpha^{(t-1)})) = \underbrace{\big(-f_i^*(-\alpha_i^t) - \lambda n g^*(v^{(t)})\big)}_{A_i} - \underbrace{\big(-f_i^*(-\alpha_i^{(t-1)}) - \lambda n g^*(v^{(t-1)})\big)}_{B_i}$$

where i is the index sampled at iteration t (see Line 3). For Algorithm IV.1, by the definition of $\alpha_i^{(t)}$ given on Lines 4 and 5 we have

$$A_i = \max_{\substack{\alpha_i \in -\mathscr{D}_{f^*} \\ \text{s.t. } \beta_i/\alpha_i \geq 1}} -f_i^*(-\alpha_i) - \lambda n g^*\Big(\frac{1}{\lambda n}(\alpha_i - \alpha_i^{(t-1)})x_i + \frac{1}{\lambda n}\sum_{j=1}^{n}\alpha_j^{(t-1)}x_j - \frac{1}{\lambda}\psi\Big).$$

Using the smoothness inequality on g^* which is 1-smooth as g is 1-strongly convex,

$$g^*(v^{(t-1)} + \Delta v) \leq h(v^{(t-1)}, \Delta v)$$
$$\text{where } h(v^{(t-1)}, \Delta v) = g^*(v^{(t-1)}) + \nabla g^*(v^{(t-1)})^\top \Delta v + \frac{1}{2}\|\Delta v\|^2.$$

Hence setting Δv to $(\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i$, we can lower bound A_i with

$$A_i \geq \max_{\substack{\alpha_i \in -\mathscr{D}_{f^*} \\ \text{s.t. } \beta_i/\alpha_i \geq 1}} -f_i^*(-\alpha_i) - \lambda n h\big(v^{(t-1)}, (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i\big).$$

For Algorithm IV.2, by definition of $\alpha_i^{(t)}$ stated at Lines 4 and 5 combined with the modified argmax relation (IV.16),

$$A_i = \max_{\substack{\alpha_i \in -\mathscr{D}_{f^*} \\ \text{s.t. } \beta_i/\alpha_i \geq 1}} -f_i^*(-\alpha_i) - \lambda n h\big(v^{(t-1)}, (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)})x_i\big).$$

As both $\alpha_i^{(t-1)}$ and α_i^* belong to $\{\alpha_i \in -\mathscr{D}_{f^*}, \beta_i/\alpha_i \geq 1\}$, for any $s_i \in [0,1]$, the convex combination $\alpha_i = (1-s_i)\alpha_i^{(t-1)} + s_i\alpha_i^*$ belongs to it as well. Hence, for both algorithms, A_i is higher than the previous quantity evaluated at this specific α_i. Namely,

$$A_i \geq -f_i^*\big(-\big((1-s_i)\alpha_i^{(t-1)} + s_i\alpha_i^*\big)\big) - \lambda n h\big(v^{(t-1)}, (\lambda n)^{-1} s_i(\alpha_i^* - \alpha_i^{(t-1)})x_i\big).$$

We then use Lemma 5, in which $-\alpha_i^* \in \mathscr{D}_{f^*}$ stands for x and $-\alpha_i^{(t-1)} \in \mathscr{D}_{f^*}$ for y:

$$(1-s_i)f_i^*(-\alpha_i^{(t-1)}) + s_i f_i^*(-\alpha_i^*) - f_i^*(-(1-s_i)\alpha_i^{(t-1)} - s_i\alpha_i^*) \geq s_i L_i \gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*)$$

where

$$\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) = \frac{1}{s_i}\log\Big(1 - s_i + s_i\frac{\alpha_i^*}{\alpha_i^{(t-1)}}\Big) - \log\frac{\alpha_i^*}{\alpha_i^{(t-1)}}.$$

This inequality is used instead of the strong convex inequality of the classic SDCA analysis [SSZ14]. If we plug this inequality into A_i we obtain

$$\begin{aligned}
A_i &\geq -s_i f^*(-\alpha_i^*) - (1-s_i)f^*(-\alpha_i^{(t-1)}) + s_i L_i \gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) \\
&\quad - \lambda n g^*(v^{(t-1)}) - s_i(\alpha_i^* - \alpha_i^{(t-1)})x_i^\top \nabla g^*(v^{(t-1)}) - \frac{s_i^2(\alpha_i^* - \alpha_i^{(t-1)})^2}{2\lambda n}\|x_i\|^2 \\
&= -s_i(f^*(-\alpha_i^*) - f^*(-\alpha_i^{(t-1)})) - f^*(-\alpha_i^{(t-1)}) - \lambda n g^*(v^{(t-1)}) \\
&\quad - s_i(\alpha_i^* - \alpha_i^{(t-1)})x_i^\top \nabla g^*(v^{(t-1)}) + s_i\Big(L_i\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) - \frac{s_i}{2\lambda n}\|x_i\|^2(\alpha_i^* - \alpha_i^{(t-1)})^2\Big).
\end{aligned}$$

Hence, we retrieve B_i and rewrite the previous inequality as

$$\begin{aligned}
s_i^{-1}(A_i - B_i) \geq -\big(f^*(-\alpha_i^*) - f^*(-\alpha_i^{(t-1)})\big) - (\alpha_i^* - \alpha_i^{(t-1)})x_i^\top \nabla g^*(v^{(t-1)}) \\
+ L_i\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) - \frac{s_i}{2\lambda n}\|x_i\|^2(\alpha_i^* - \alpha_i^{(t-1)})^2.
\end{aligned}$$

We can sum over all possible sampled i and weight each entry with s_i^{-1} to obtain

$$\begin{aligned}
\sum_{i=1}^n s_i^{-1}(A_i - B_i) \geq -\sum_{i=1}^n \Big(f^*(-\alpha_i^*) - f^*(-\alpha_i^{(t-1)})\Big) - \Big\langle \nabla g^*(v^{(t-1)}) \,\Big|\, \sum_{i=1}^n (\alpha_i^* - \alpha_i^{(t-1)})x_i \Big\rangle \\
+ \sum_{i=1}^n \Big(L_i\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) - \frac{s_i}{2\lambda n}\|x_i\|^2(\alpha_i^* - \alpha_i^{(t-1)})^2\Big). \qquad \text{(IV.26)}
\end{aligned}$$

Then since g^* is convex, we obtain

$$\begin{aligned}
\Big\langle \nabla g^*(v^{(t-1)}) \,\Big|\, \sum_{i=1}^n (\alpha_i^* - \alpha_i^{(t-1)})x_i \Big\rangle &= \big\langle \nabla g^*(v^{(t-1)}) \mid \lambda n(v(\alpha^*) - v^{(t-1)}) \big\rangle \\
&\leq \lambda n\big(g^*(v(\alpha^*)) - g^*(v^{(t-1)})\big),
\end{aligned}$$

which can be injected in Equation (IV.26) leading to

$$\sum_{i=1}^{n} s_i^{-1}(A_i - B_i) \geq -\sum_{i=1}^{n}\left(f^*(-\alpha_i^*) - f^*(-\alpha_i^{(t-1)})\right) + \lambda n g^*(v(\alpha^*)) - \lambda n g^*(v^{(t-1)}) + \sum_{i=1}^{n}\left(L_i\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) - \frac{s_i}{2\lambda n}\|x_i\|^2(\alpha_i^* - \alpha_i^{(t-1)})^2\right).$$

Finally, since $A_i - B_i = n(D(\alpha^{(t,i)}) - D(\alpha^{(t-1)}))$, we obtain

$$\sum_{i=1}^{n} s_i^{-1}\left(D(\alpha^{(t,i)}) - D(\alpha^{(t-1)})\right) \geq D(\alpha^*) - D(\alpha^{(t-1)}) + \frac{1}{n}\sum_{i=1}^{n}\left(L_i\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) - \frac{s_i}{2\lambda n}\|x_i\|^2(\alpha_i^* - \alpha_i^{(t-1)})^2\right).$$

This concludes the proof of Lemma 8. ■

From Lemma 8, we obtain a contraction speed as soon as $G(s, \alpha^{(t-1)}, \alpha^*) \geq 0$. If $\alpha_i^{(t-1)} \neq \alpha_i^*$ this is obtained if

$$\forall i \in \{1, \dots, n\}, \quad L_i\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*) - \frac{s_i}{2\lambda n}\|x_i\|^2(\alpha_i^* - \alpha_i^{(t-1)})^2 \geq 0 \tag{IV.27}$$

$$\Leftrightarrow \quad \forall i \in \{1, \dots, n\}, \quad \frac{\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*)}{\left(1 - \frac{\alpha_i^{(t-1)}}{\alpha_i^*}\right)^2} - s_i\frac{\|x_i\|^2\alpha_i^{*2}}{2\lambda n L_i} \geq 0.$$

By definition of γ we have

$$\frac{\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*)}{\left(1 - \frac{\alpha_i^{(t-1)}}{\alpha_i^*}\right)^2} = \frac{\log\left(1 - s_i + s_i\frac{\alpha_i^*}{\alpha_i^{(t-1)}}\right) - s_i\log\frac{\alpha_i^*}{\alpha_i^{(t-1)}}}{s_i\left(1 - \frac{\alpha_i^{(t-1)}}{\alpha_i^*}\right)^2}.$$

As $\alpha_i^{(t-1)}/\alpha_i^*$ is bounded by β_i/α_i^*, we apply Lemma 6 to obtain

$$\frac{\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*)}{\left(1 - \frac{\alpha_i^{(t-1)}}{\alpha_i^*}\right)^2} \geq \frac{\log\left(1 - s_i + s_i\frac{\alpha_i^*}{\beta_i}\right) - s_i\log\frac{\alpha_i^*}{\beta_i}}{s_i\left(1 - \frac{\beta_i}{\alpha_i^*}\right)^2},$$

and as $\beta_i/\alpha_i^* \geq 1$, we can apply Lemma 7 leading to

$$\frac{\gamma(s_i, \alpha_i^{(t-1)}, \alpha_i^*)}{\left(1 - \frac{\alpha_i^{(t-1)}}{\alpha_i^*}\right)^2} \geq \frac{(1 - s_i)\left(\frac{\alpha_i^*}{\beta_i} - 1 + \log\frac{\beta_i}{\alpha_i^*}\right)}{\left(1 - \frac{\beta_i}{\alpha_i^*}\right)^2}.$$

Finally the convergence condition from Equation (IV.27) is satisfied when

$$\forall i \in \{1,\dots,n\}, \quad \left(1-s_i\right)\left(\frac{\alpha_i^*}{\beta_i} - 1 + \log\frac{\beta_i}{\alpha_i^*}\right) - s_i \frac{\|x_i\|^2 \alpha_i^{*2}}{2\lambda n L_i}\left(1-\frac{\beta_i}{\alpha_i^*}\right)^2 \geq 0$$

which is true for any $s_i \in [0, \sigma_i]$ where

$$\sigma_i = \left(1 + \frac{\|x_i\|^2 \alpha_i^{*2}}{2\lambda n L_i} \frac{\left(1-\frac{\beta_i}{\alpha_i^*}\right)^2}{\frac{\alpha_i^*}{\beta_i} + \log\frac{\beta_i}{\alpha_i^*} - 1}\right)^{-1}.$$

Theorem 1 is obtained by sampling uniformly i, meaning haing all s_i equal. Hence, to fulfill Equation (IV.27), we set

$$s_i = \min_{j\in\{1,\dots,n\}} \sigma_j \tag{IV.28}$$

for all $i \in \{1,\dots,n\}$. We then lower bound the expectation of $D(\alpha^{(t)}) - D(\alpha^{(t-1)})$ over all possible sampled i and obtain

$$\mathbb{E}[D(\alpha^{(t)}) - D(\alpha^{(t-1)})] = \frac{1}{n}\sum_{i=1}^{n} D(\alpha^{(t,i)}) - D(\alpha^{(t-1)}) \geq \frac{\min_j \sigma_j}{n}(D(\alpha^*) - D(\alpha^{(t-1)})),$$

by multiplying Equation (IV.25) with $\min_{j\in\{1,\dots,n\}} \sigma_j / n$ and removing the quantity $G^{(t-1)} \geq 0$. This leads to the following convergence speed after t iterations,

$$\mathbb{E}[D(\alpha^*) - D(\alpha^{(t)})] \leq \left(1 - \frac{\min_j \sigma_j}{n}\right)^t (D(\alpha^*) - D(\alpha^{(0)})),$$

which concludes the proof of Theorem 1. ■

7.8 Proof of Theorem 2

Instead of taking all s_i equal as in the uniform sampling setting (see Equation (IV.28)), we rather parametrize s_i by $\frac{\bar\sigma}{\rho_i n}$ where ρ_i is the probability of sampling i. Then, we obtain the following expectation under ρ,

$$\mathbb{E}_\rho[D(\alpha^{(t)}) - D(\alpha^{(t-1)})] = \sum_{i=1}^{n} \rho_i D(\alpha^{(t,i)}) - D(\alpha^{(t-1)}).$$

Since we have $\rho_i = \frac{n}{\bar\sigma} s_i^{-1}$ we obtain the following inequality using Lemma 8:

$$\mathbb{E}_\rho[D(\alpha^{(t)}) - D(\alpha^{(t-1)})] \geq \frac{\bar\sigma}{n}\Big(D(\alpha^*) - D(\alpha^{(t-1)}) + G\big(\bar\sigma(\rho n)^{-1}, \alpha^{(t-1)}, \alpha^*\big)\Big). \tag{IV.29}$$

To ensure that $G\big(\bar{\sigma}(\rho n)^{-1}, \alpha^{(t-1)}, \alpha^*\big) \geq 0$ while keeping the biggest gain, we must satisfy the constraint from Equation (IV.27) and find feasible ρ and $\bar{\sigma}$ that maximize the following problem:

$$\max_{\bar{\sigma}} \bar{\sigma} \quad \text{subject to} \quad \frac{\bar{\sigma}}{\rho_i n} \in [0, \sigma_i], \quad \rho_i \geq 0, \quad \sum_{i=1}^{n} \rho_i = 1.$$

This problem is solved by Proposition 1 of [ZZ15] and leads to the following choices:

$$\rho_i = \frac{\sigma_i^{-1}}{\sum_{j=1}^{n} \sigma_j^{-1}} \quad \text{and} \quad \bar{\sigma} = \Big(\frac{1}{n} \sum_{i=1}^{n} \sigma_i^{-1}\Big)^{-1}.$$

This choice for ρ and $\bar{\sigma}$ ensures $G(\bar{\sigma}(\rho n)^{-1}, \alpha^{(t-1)}, \alpha^*) \geq 0$ hence Equation (IV.29) without $G\big(\bar{\sigma}(\rho n)^{-1}, \alpha^{(t-1)}, \alpha^*\big)$ leads to

$$\mathbb{E}_\rho[D(\alpha^{(t)}) - D(\alpha^{(t-1)})] \geq \frac{\bar{\sigma}}{n}(D(\alpha^*) - D(\alpha^{(t-1)})),$$

and finally, after t iterations, we have

$$\mathbb{E}[D(\alpha^*) - D(\alpha^{(t)})] \leq \Big(1 - \frac{\bar{\sigma}}{n}\Big)^t (D(\alpha^*) - D(\alpha^{(0)})),$$

which concludes the proof of Theorem 2. ■

7.9 Proof of Proposition 4

With $f_i^*(-\alpha_i) = -y_i - y_i \log \frac{\alpha_i}{y_i}$, let

$$\phi(\alpha_i) = y_i + y_i \log \frac{\alpha_i}{y_i} - \frac{\lambda n}{2} \Big\| w^{(t-1)} + (\lambda n)^{-1}(\alpha_i - \alpha_i^{(t-1)}) x_i \Big\|^2$$

be the function to optimize. Note that ϕ is a concave function from $-\mathscr{D}_{f^*}$ to $\mathbb{R}$ and hence it reaches its minimum if its gradient is zero:

$$\phi'(\alpha_i) = \frac{y_i}{\alpha_i} - w^{(t-1)\top} x_i - \frac{\|x_i\|^2}{\lambda n}(\alpha_i - \alpha_i^{(t-1)}) = 0.$$

This second order equation in α_i has a unique positive solution, the one stated in Proposition 4.

7.10 Proof of Proposition 5

Given that we are using Ridge regularization, the values of f_i^* and g^* are

$$f_i^*(v) = -y_i - y_i \log\left(\frac{-v}{y_i}\right) \quad \text{and} \quad g^*(w) = g(w) = \frac{1}{2}\|w\|^2.$$

Hence the conditions at optimum (IV.5) and (IV.14) become

$$w^* = \frac{1}{\lambda n}\sum_{i=1}^{n} \alpha_i^* x_i - \frac{1}{\lambda}\psi \quad \text{and} \quad \forall i \in \{1,\dots n\}, \quad \alpha_i^* = \frac{y_i}{w^{*\top} x_i}. \tag{IV.30}$$

By combining both equations with Equation (IV.30), we have

$$\forall i \in \{1,\dots n\}, \quad \alpha_i^* = \frac{\lambda n y_i}{\sum_{j=1}^n \alpha_j^* x_j^\top x_i - n\psi^\top x_i}.$$

Since the inner products $x_i^\top x_j$ and α_i are non-negative, we can remove the terms $\sum_{j\neq i} \alpha_j^* x_j^\top x_i$ and upper bound the dual variable with

$$\forall i \in \{1,\dots n\}, \quad \alpha_i^* \leq \frac{\lambda n y_i}{\alpha_i^* \|x_i\|^2 - n\psi^\top x_i}.$$

By solving this second order inequality, we can derive the following upper bound for all α_i^*:

$$\alpha_i^* \leq \frac{1}{2\|x_i\|^2}\left(n\psi^\top x_i + \sqrt{(n\psi^\top x_i)^2 + 4\lambda n y_i \|x_i\|^2}\right),$$

which concludes the proof. ■

7.11 Proof of Remark 2

At each iteration the closed form solution is given by Proposition 4:

$$\alpha_i^t = \frac{1}{2}\left(\sqrt{\left(\alpha_i^{(t-1)} - \frac{\lambda n}{\|x_i\|^2} w^{(t-1)\top} x_i\right)^2 + 4\lambda n \frac{y_i}{\|x_i\|^2}} + \alpha_i^{(t-1)} - \frac{\lambda n}{\|x_i\|^2} w^{(t-1)\top} x_i\right).$$

Since the inner products $x_i^\top x_j$ are non-negative, we obtain

$$\alpha_i^{(t-1)} - \frac{\lambda n}{\|x_i\|^2} w^{(t-1)\top} x_i = n\frac{\psi^\top x_i}{\|x_i\|^2} - \sum_{j\neq i} \alpha_j^{(t-1)} \frac{x_j^\top x_i}{\|x_i\|^2} \leq n\frac{\psi^\top x_i}{\|x_i\|^2}$$

and since α_i^t is increasing with $(\alpha_i^{(t-1)} - \frac{\lambda n}{\|x_i\|^2} w^{(t-1)\top} x_i)$, we obtain

$$\alpha_i^t \leq \frac{1}{2}\left(\sqrt{n\frac{\psi^\top x_i^2}{\|x_i\|^4} + 4\lambda n \frac{y_i}{\|x_i\|^2}} + n\frac{\psi^\top x_i}{\|x_i\|^2}\right) = \beta_i,$$

which concludes the proof. ■

7.12 Proof of Proposition 6

This proposition easily follows from the following computation

$$\begin{aligned}\zeta^* &= \operatorname*{arg\,max}_{\zeta\in(0,+\infty)^n} \frac{1}{n}\sum_{i=1}^{n} -y_i - y_i \log\frac{\zeta_i}{y_i} - \lambda g^*\Big(\frac{1}{\lambda n}\sum_{i=1}^{n}\zeta_i\xi_i - \frac{1}{\lambda}\psi\Big)\\ &= \operatorname*{arg\,max}_{\zeta\in(0,+\infty)^n} \frac{1}{n}\sum_{i=1}^{n} -y_i - y_i \log\frac{\zeta_i}{y_i} + y_i\log(c_i) - \lambda g^*\Big(\frac{1}{\lambda n}\sum_{i=1}^{n} c_i\zeta_i x_i - \frac{1}{\lambda}\psi\Big)\\ &= \operatorname*{arg\,max}_{\zeta\in(0,+\infty)^n} D(c\cdot\zeta),\end{aligned}$$

where D is the original dual problem and $c\cdot\zeta$ is the element wise product of the vectors c and ζ. Then, since

$$\operatorname*{arg\,max}_x\{x\mapsto f(cx)\} = \frac{1}{c}\operatorname*{arg\,max}\{x\mapsto f(x)\},$$

which remains valid in the multivariate case, we obtain $\zeta_i^* = \alpha_i^*/c_i$ for any $i = 1,\dots,n$. ■

7.13 Proof of Proposition 7

Using $\alpha^* = \bar{\alpha}\kappa$ the dual problem becomes one dimensional

$$D(\bar{\alpha}) = \frac{1}{n}\sum_{i=1}^{n} y_i + y_i\log\tfrac{\kappa_i\bar{\alpha}}{y_i} - \frac{\lambda}{2}\Big\|\frac{1}{\lambda n}\sum_{i=1}^{n}\kappa_i\bar{\alpha}x_i - \frac{1}{\lambda}\psi\Big\|^2.$$

This problem is concave in $\bar{\alpha}$ and the optimal $\bar{\alpha}$ is obtained by setting the derivative to zero:

$$D'(\bar{\alpha}) = \frac{1}{n}\sum_{i=1}^{n}\frac{y_i}{\bar{\alpha}} - \Big\langle\frac{1}{\lambda n}\bar{\alpha}\sum_{i=1}^{n}\kappa_i x_i - \frac{1}{\lambda}\psi \,\Big|\, \frac{1}{n}\sum_{i=1}^{n}\kappa_i x_i\Big\rangle = 0.$$

This leads to the following second order equation

$$\Big\|\frac{1}{n}\sum_{i=1}^{n}\kappa_i x_i\Big\|^2\bar{\alpha}^2 - \Big\langle\psi \,\Big|\, \frac{1}{n}\sum_{i=1}^{n}\kappa_i x_i\Big\rangle\bar{\alpha} - \frac{\lambda}{n}\sum_{i=1}^{n} y_i = 0,$$

which has a unique positive solution

$$\bar{\alpha} = \frac{1}{2\big\|\frac{1}{n}\sum_{i=1}^{n}\kappa_i x_i\big\|^2}\left(\Big\langle\psi \,\Big|\, \frac{1}{n}\sum_{i=1}^{n}\kappa_i x_i\Big\rangle + \sqrt{\Big\langle\psi \,\Big|\, \frac{1}{n}\sum_{i=1}^{n}\kappa_i x_i\Big\rangle^2 + 4\frac{\lambda}{n}\sum_{i=1}^{n} y_i\Big\|\frac{1}{n}\sum_{i=1}^{n}\kappa_i x_i\Big\|^2}\right),$$

and concludes the proof. ■

CHAPTER V

tick: a Python library for statistical learning, with an emphasis on Hawkes processes and time-dependent models

Abstract

This chapter introduces `tick`, is a statistical learning library for Python 3, with a particular emphasis on time-dependent models, such as point processes, tools for generalized linear models and survival analysis. The core of the library provides model computational classes, solvers and proximal operators for regularization. It relies on a C++ implementation and state-of-the-art optimization algorithms to provide very fast computations in a single node multi-core setting. Source code and documentation can be downloaded from https://github.com/X-DataInitiative/tick.
Keywords. Statistical Learning; Python; Hawkes processes; Optimization; Generalized linear models; Point Process; Survival Analysis

1 Introduction

The aim of the `tick` library is to provide for the Python community a large set of tools for statistical learning, previously not available in any framework. Though `tick` focuses on time-dependent modeling, it actually introduces a set of tools that go way beyond this particular set of models, thanks to a highly modular optimization toolbox. It benefits from thorough documentation (including tutorials with many examples), and a strongly tested Python API that brings to the scientific community cutting-edge algorithms with a high level of customization. Optimization algorithms

such as SVRG [JZ13] or SDCA [SSZ13] are among the several optimization algorithms available in `tick` that can be applied (in a modular way) to a large variety of models. An emphasis is placed on time-dependent models: from the Cox regression model [ABGK12], a very popular model in survival analysis, to Hawkes processes, used in a wide range of applications such as geophysics [Oga88], finance [BMM15] and more recently social networks [ZZS13a, XFZ16]. To the best of our knowledge, `tick` is the most comprehensive library that deals with Hawkes processes, since it brings parametric and nonparametric estimators of these models to a new accessibility level.

2 Existing Libraries

The `tick` library follows, whenever possible, `scikit-learn`'s API [PVG+11, BLB+13] which is well-known for its completeness and ease of use. However, while `scikit-learn` targets a wide spectrum, `tick` has a more specific objective: implementing highly-optimized algorithms with a particular emphasis on time-dependent modeling (not proposed in `scikit-learn`). The `tick` optimization toolbox relies on state-of-the-art optimization algorithms, and is implemented in a very modular way. It allows more possibilities than other `scikit-learn` API based optimization libraries such as `lightning`[1].

A wide variety of time-dependent models are taken care of by `tick`, which makes it the most comprehensive library that deals with Hawkes processes for instance, by including the main inference algorithms from the literature. Despite the growing interest in Hawkes models, very few open source packages are available. There are mainly three of them. The library `pyhawkes`[2] proposes a small set of Bayesian inference algorithms for Hawkes processes. `hawkes` R[3] is an R-based library that provides a single estimation algorithm, and is hardly optimized. Finally, `PtPack`[4] a C++ library which proposes parametric maximum likelihood estimators, with sparsity-inducing regularizations. However, since `tick` is a Python library, it addresses a different community to PtPack. Moreover, as illustrated below, `tick` provides better performance than `PtPack`.

3 Package Architecture

The `tick` library has four main modules: `tick.hawkes` for Hawkes processes (see Section 4 for a detailed review), `tick.linear_model` with linear, logistic and Poisson

[1] http://contrib.scikit-learn.org/lightning
[2] https://github.com/slinderman/pyhawkes
[3] https://cran.r-project.org/web/packages/hawkes/hawkes.pdf
[4] https://github.com/dunan/MultiVariatePointProcess

Table V.1: tick allows the user to combine many models, prox and solvers

Model	Proximal operator	Solver
Linear regression	SLOPE	Gradient Descent
Logistic regression	L1 (Lasso)	Stochastic Variance Reduced Gradient
Poisson regression	Total Variation	Stochastic Gradient Descent
Cox regression	Group L1	Accelerated Gradient Descent
Hawkes with exp. kernels	L2 (Ridge)	Stochastic Dual Coordinate Ascent

regression, `tick.robust` for robust linear models and `tick.survival` for survival analysis. Each of these modules provide simulation tools and learners to easily learn from data. The core of `tick` is made of easy to combine penalization techniques (`tick.prox` module) and several convex solvers (`tick.solver`), to train almost any available model in the library, see Table V.1 for a non-exhaustive list of possible combinations. The full structure of the `tick` library is detailed in Figure V.1. Here is a short description of the contents of each module:

- `tick.hawkes` : Inference and simulation of Hawkes processes, with both parametric and non-parametric estimation techniques and flexible tools for simulation. It is split in three submodules: `tick.hawkes.inference`, `tick.hawkes.simulation`, `tick.hawkes.model`.
- `tick.linear_model` : Inference and simulation of linear models, including among others linear, logistic and Poisson regression, with a large set of penalization techniques and solvers.
- `tick.robust` : Tools for robust inference. It features tools for outliers detection and models such as Huber regression, among others robust losses.
- `tick.survival` : Inference and simulation for survival analysis, including Cox regression with several penalizations.
- `tick.prox` : Proximal operators for penalization of model weights. Such an operator can be used with (almost) any model and any solver.
- `tick.solver` : A module that provides a bunch of state-of-the-art optimization algorithms, including both batch and stochastic solvers.
- `tick.dataset` : Provides easy access to datasets used as benchmarks in tick.
- `tick.plot` : Some plotting utilities used in tick, such as plots for point processes and solver convergence.

Figure V.1: Structure of the `tick` library

Table V.2: Models and estimation techniques for Hawkes processes available in `tick`

Non Parametric	Parametric
EM [LM11]	Single exponential kernel
Basis kernels [ZZS13a]	Sum of exponentials kernels
Wiener-Hopf [BM14]	Sum of gaussians kernels [XFZ16]
NPHC [ABG+17]	ADM4 [ZZS13a]

4 Hawkes

A Hawkes process [HO74] is a multivariate point-process. Namely, it models timestamps, also called ticks $\{t_k^i\}_{i\geq 1}$ of nodes $i = 1, \dots, D$ using a multivariate counting process $N_t = [N_t^1 \cdots N_t^D]$, for $t \geq 0$, where $N_t^i = \sum_{k\geq 1} \mathbb{1}_{t_k^i \leq t}$ for any $t \geq 0$. In a Hawkes process the intensity of component N^i has the following auto-regressive structure:

$$\lambda_i(t) = \mu_i + \sum_{j=1}^{D} \int \phi_{ij}(t-s)\, \mathrm{d}N_j(s) = \mu_i + \sum_{j=1}^{D} \sum_{k:t_k^j < t} \phi_{ij}(t - t_k^j).$$

The $\mu_i \geq 0$ are called *baselines* intensities, and correspond to the exogenous intensity of events from node i, and ϕ_{ij} for $1 \leq i, j \leq D$ are called *kernels*, that quantify the influence of past events from node j on the intensity of events from node i. The main parametric model for the kernels is the so-called *exponential* kernel, in which we consider $\phi_{ij}(t) = \alpha_{ij}\beta \exp(-\beta t)$ for $\alpha_{ij} > 0$ and $\beta > 0$. In this model $A = [\alpha_{i,j}]_{1\leq i,j\leq d}$ is understood as an *adjacency matrix*, since entry $A_{i,j} \geq 0$ quantifies the impact of the activity of node j on the activity of node i, while $\beta > 0$ is a memory parameter.

Distributing a comprehensive open source library for Hawkes processes is one of the primary aims of the `tick` library: it provides many non-parametric and parametric estimation algorithms that are listed in Table V.2. This diversity of algorithms is illustrated in Figure V.2 in which we show how two kernels of different shapes are estimated by four different algorithms. A first use case for modeling high-frequency financial data is given in Section 6.2, while a second use-case about propagation analysis of earthquake aftershocks can be found in Figure V.3.

5 Benchmarks

While `tick` is a Python library, all the heavy computations run in C++ which communicates with Python with SWIG (Simplified Wrapper and Interface Generator) [BFK+96]. Thanks to SWIG, the Python objects have a very easy access to full C++ objects that share their memory and work on the same dataset without needing any

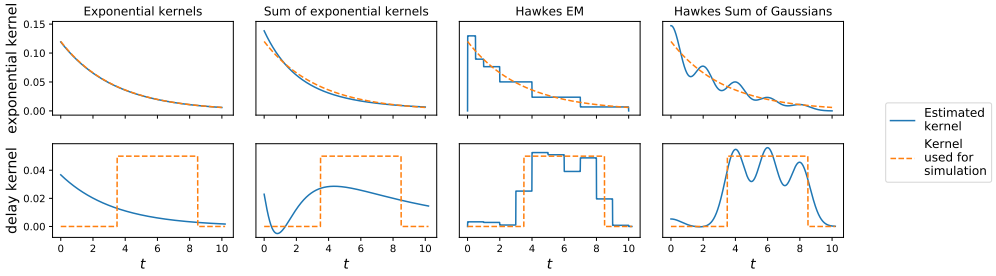

Figure V.2: Illustration of different kernel shapes and estimations obtained by `tick` on two 1-dimensional simulated Hawkes processes.

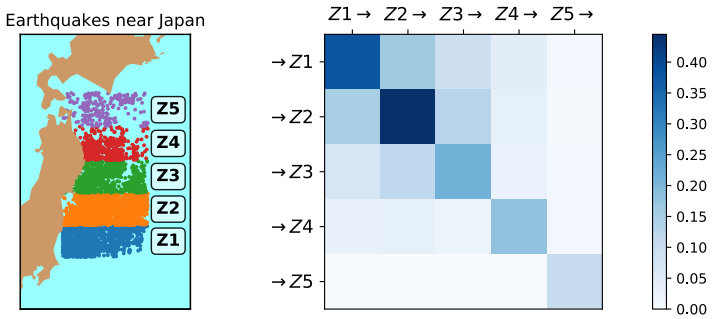

Figure V.3: Modeling of earthquake propagation with Hawkes processes on a dataset from [Oga88]. The left hand side gives the location of the earthquakes while right hand side illustrates the propagation matrix, namely how likely an earthquake in a given zone will trigger an aftershock in another zone.

copy. This is particularly useful for optimization toolbox where `model`, `prox` and `solver` are symbolically linked in Python and then run fully in C++. Also, the C++ part of the library is independent and with some effort is usable without the Python part. This allows the developers to analyze the code with any profiling tool compatible with C++ and hence produce code optimized in depth. For all these reasons, `tick` is a very fast library that we compare to `scikit-learn`, `hawkes` R and `PtPack` in the following benchmarks. In Figure V.4, we benchmark computational times for both simulation and estimation of Hawkes processes (with exponential kernels) using `hawkes` R (where only simulation is available), `PtPack` and `tick` on respectively 2, 4 and 16 cores. The model fits in plots "Fit" and "Multicore fit" are compared on simulated 16-dimensional Hawkes processes, with an increasing number of events: small=5×10^4, medium=2×10^5, large=10^6, xlarge=5×10^7, while 200, 400 and 750 dimensional Hawkes processes are fitted in plot "High-dimensional fitting". We observe that `tick` outperforms by several orders of magnitudes both `hawkes` R and `PtPack`, in

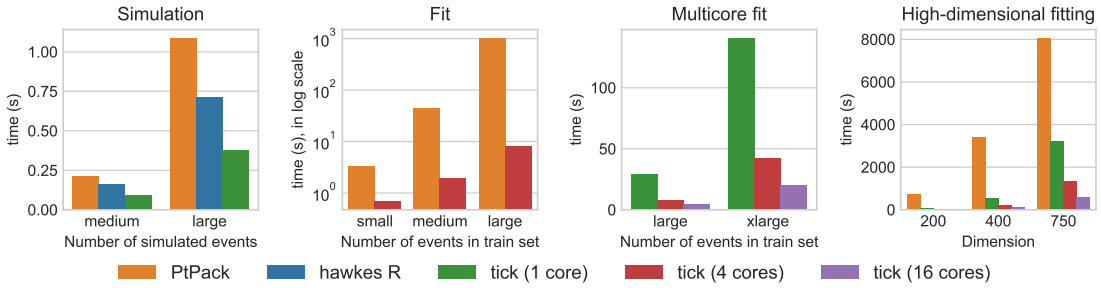

Figure V.4: Computational timings of `tick` versus `PtPack` and `hawkes` R. `tick` strongly outperforms both libraries for simulation and fitting (note that the "Fit" graph is in log-scale). "Multicore fit" and "High-dimensional fitting" plots show that `tick` benefits from multi-core environments to speed up computations.

dataset	# samples	# features	density
IJCNN	141,691	22	100 %
Covtype	581,012	54	100 %
Adult	32,561	123	11.3 %
RCV1-ccat	804,414	47,236	0.0016 %
URL	2,396,130	3,231,961	0.000036 %
KDD 2010	19,264,097	1,163,024	0.00078 %

Figure V.5: Datasets used to perform binary logistic regression.

particular for large datasets. We also compare computational timings to fit a logistic regression with `tick` and `scikit-learn`. These experiments are run on commonly used datasets (including large scale ones) described in Table V.5. Note that Covtype has been standardized, hence the first two datasets IJCNN and Covtype are dense and the last four datasets are sparse. Two types of penalization have been tested: ℓ_1 (Lasso) and ℓ_2 (Ridge), since these are the only ones proposed by `scikit-learn`. In both cases the regularization parameter λ has been set to $1/n$ where n is the number of samples, and we consider the default step-sizes proposed by both libraries. For a fair comparison, we considered the SAGA algorithm, see [DBLJ14], to minimize the penalized logistic regression objective, since both libraries implement it. Results are given in Figure V.6. Overall, `tick` is much faster since it makes faster iterations: both libraries reach roughly the same objective after each pass over the data (since both of them use the same algorithm, only their implementations differ). Moreover, ℓ_1 penalization in high dimension is difficult for `scikit-learn` (see URL and KDD 2010) whereas `tick` handles it without any additional problem.

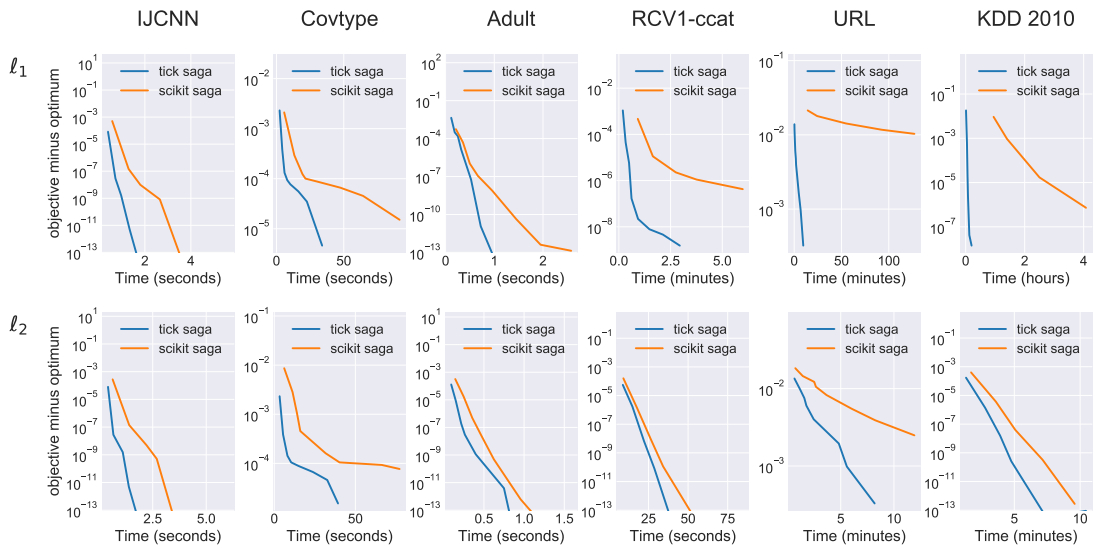

Figure V.6: Speed comparison with `scikit-learn` library. These plots display time needed to achieve a given precision for logistic regression with ℓ_1 and ℓ_2 penalizations on commonly used datasets. In both cases we use SAGA solver as the two libraries provide it.

6 Examples

In this section we provide examples of `tick` usage either to analyze timestamps with Hawkes processes or to run convex optimization experiments and comparisons. These examples are picked from a gallery of the documentation page [5].

6.1 Estimate Hawkes intensity

The `tick` library provides many utilities to easily work with Hawkes processes. This includes tools to plot the kernels (see Figure V.2) or their norms (the $D \times D$ matrix such that $(\Phi)_{ij} = \int_0^\infty \phi_{ij}(t)dt$) to describe the joint dynamics of the events series or as presented here, tools to plot the point process events with their associated intensity. In this example we simulate timestamps from Hawkes process with exponential kernels. Then, we estimate the kernel parameters from the generated timestamps and rereate the intensity from the timestamps and the estimated kernels. In Figure V.7 we see that the estimated intensity is very close from the generating one.

[5] https://x-datainitiative.github.io/tick/auto_examples

```
from tick.hawkes import SimuHawkesSumExpKernels, HawkesSumExpKern
from tick.plot import plot_point_process

decays = [0.1, 0.5, 1.]
baseline = [0.1]
adjacency = [[[0.4, .1, .2]]]

hawkes_exp_kernels = SimuHawkesSumExpKernels(
    adjacency=adjacency, decays=decays, baseline=baseline,
    end_time=2000)

hawkes_exp_kernels.track_intensity(0.1)
hawkes_exp_kernels.simulate()

learner = HawkesSumExpKern(decays, penalty='elasticnet')
learner.fit(hawkes_exp_kernels.timestamps)

learner.plot_estimated_intensity(hawkes_exp_kernels.timestamps,
                                 t_min=100, t_max=200)

plot_point_process(hawkes_exp_kernels, plot_intensity=True,
                   t_min=100, t_max=200)
```

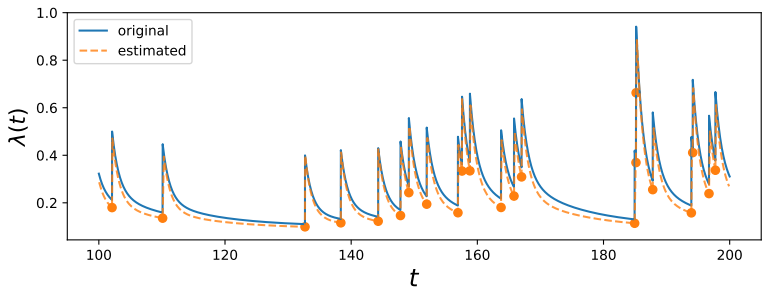

Figure V.7: Original and retrieved intensities of a Hawkes process.

6.2 Fit Hawkes on finance data

This example fits Hawkes kernels on finance data provided by tick-datasets repository[6]. It show in Figure V.8 the kernel norms of a Hawkes process fitted on finance data of Bund market place. This reproduces experiments run in [BJM16]. P_u (resp. P_d) counts the number of upward (resp. downward) mid-price moves and T_A (resp.

[6] https://github.com/X-DataInitiative/tick-datasets/tree/master/hawkes/bund

T_b) counts the number of market orders at the ask (resp. bid) that do not move the price. We observe expected behavior with for example mid-price moving downward triggering (resp. preventing) market orders at the ask (resp. at the bid).

```
from tick.dataset import fetch_hawkes_bund_data
from tick.hawkes import HawkesConditionalLaw
from tick.plot import plot_hawkes_kernel_norms

timestamps_list = fetch_hawkes_bund_data()

kernel_discretization = np.hstack((0, np.logspace(-5, 0, 50)))
hawkes_learner = HawkesConditionalLaw(
    claw_method="log", delta_lag=0.1, min_lag=5e-4, max_lag=500,
    quad_method="log", n_quad=10, min_support=1e-4, max_support=1,
    n_threads=4)

hawkes_learner.fit(timestamps_list)

plot_hawkes_kernel_norms(hawkes_learner,
                         node_names=["P_u", "P_d", "T_a", "T_b"])
```

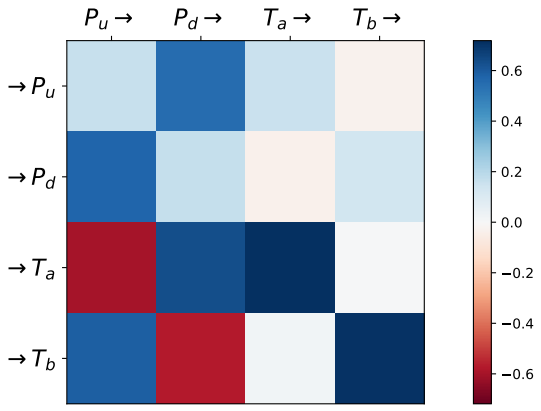

Figure V.8: Kernel norms of a Hawkes process fitted on finance data.

6.3 SVRG with an adaptive step size

The `tick` code base is very convenient to implement new algorithms. It is very modular and once a new brick is implemented in the optimization framework it can

be used with any other bricks. Also, it is very easy to implement and test a new algorithm. For example this code uses a variation of SVRG algorithm [JZ13] (see Section 4 of Chapter III) that chooses an appropriate step size while running the algorithm using the Barzilai-Borwein method [BB88]. This method consists in an adaptive method to choose the step size η based on quasi Newton analysis. Such optimizers uses the following quadratic approximation,

$$f_{\text{quad}}(w) = f(w^t) + (w - w^t)^\top \nabla f(w^t) + \tfrac{1}{2}(w - w^t)^\top H^t (w - w^t),$$

where the matrix H^t is close to the hessian matrix $\nabla^2 f(w^t)$ to mimic the second order Taylor decomposition of $f(w^{t+1})$. The vector w^{t+1} that maximizes this quadratic approximation of f, is

$$w^{t+1} = w^t - \left(H^t\right)^{-1} \nabla f(w^t).$$

This update is the typical iteration of quasi-Newton methods which differs from one another by their approximation of $(H^t)^{-1}$. While the standard Newton method uses $(\nabla^2 f(w^t))^{-1}$, quasi-Newton methods find matrices that are cheaper to compute and have useful properties such as verifying the *secant condition*

$$w^t - w^{t-1} = \left(H^t\right)^{-1} \left(\nabla f(w^t) - \nabla f(w^{t-1})\right).$$

This condition is obtained by ensuring that the gradient of f_{quad} coincides with the gradient of f at w^{t-1}. This equation defines $\left(H^t\right)^{-1}$ as the solution of a linear system that might be expensive to compute. Hence, Barzilai and Borwein suggest to parametrize $\left(H^t\right)^{-1}$ as $\eta^t \boldsymbol{I}_d$, where $\boldsymbol{I}_d$ is the identity matrix. As no η^t satisfies the secant equation, they rather minimize the residuals, namely

$$\|(1/\eta^t)(w^t - w^{t-1}) - \nabla f(w^{t+1}) - \nabla f(w^t)\|^2$$

which sets

$$\eta^t = \frac{\|w^t - w^{t-1}\|^2}{(w^t - w^{t-1})^\top (\nabla f(w^{t+1}) - \nabla f(w^t))}.$$

Hence, the step is adaptive and is tunes automatically across iterations. This technique is adapted in [TMDQ16] on the SVRG algorithm that we describe in Algorithm V.1. The step is set every m iteration by using the full gradients $\widetilde{\mu}_k$ that are already computed to perform the variance reduction.

In the following Python code, we compare this method with the traditional SVRG on a toy example in which we can obtain the theoretically optimal step size. In Figure V.9 we display the convergence speeds of SVRG and SVRG-BB for several step sizes. We observe that SVRG converges very slowly if the step size is too small and does not converge at all if the step size is too big. On the contrary, SVRG-BB performs well in all cases. Hence, this adaptive solver is way less sensitive to the step size initially provided.

Algorithm V.1 SVRG-BB

Require: Starting point w^0, a step size $\eta_0 > 0$
  $k \leftarrow 0$
  for $t = 0, 1, \ldots$ **do**
    if t is a multiple of m **then**
      $\widetilde{w}_k \leftarrow w^t$
      $\widetilde{\mu}_k \leftarrow \nabla f(\widetilde{w}_k)$
      if $k > 0$ **then**
        $\eta_{k+1} \leftarrow \|\widetilde{w}_k - \widetilde{w}_{k-1}\|^2 / (\widetilde{w}_k - \widetilde{w}_{k-1})^\top (\widetilde{\mu}_k - \widetilde{\mu}_{k-1})$
      end if
      $k \leftarrow k + 1$
    end if
    Sample at random i in $\{1, \ldots, n\}$
    $w^{t+1} \leftarrow \mathrm{prox}_{\eta_k g}\Big(w^t - \eta_k\big(\nabla f_i(w^t) - \nabla f_i(\widetilde{w}_k) + \widetilde{\mu}_k\big)\Big)$
  end for

```python
from tick.simulation import weights_sparse_gauss
from tick.solver import SVRG
from tick.linear_model import SimuLogReg, ModelLogReg
from tick.prox import ProxElasticNet
from tick.plot import plot_history

n_samples, n_features, = 5000, 50
weights0 = weights_sparse_gauss(n_features, nnz=10)
intercept0 = 0.2
X, y = SimuLogReg(weights=weights0, intercept=intercept0,
                  n_samples=n_samples).simulate()

model = ModelLogReg(fit_intercept=True).fit(X, y)
prox = ProxElasticNet(strength=1e-3, ratio=0.5, range=(0, n_features))

optimal_step = 1 / model.get_lip_max()
tested_steps = [optimal_step, 1e-2 * optimal_step, 10 * optimal_step]

solvers = []
for step in tested_steps:
    svrg = SVRG(max_iter=30, tol=1e-10, verbose=False)
    svrg.set_model(model).set_prox(prox)
    svrg.solve(step=step)

    svrg_bb = SVRG(max_iter=30, tol=1e-10, verbose=False, step_type='bb')
    svrg_bb.set_model(model).set_prox(prox)
    svrg_bb.solve(step=step)

    solvers += [svrg, svrg_bb]
```

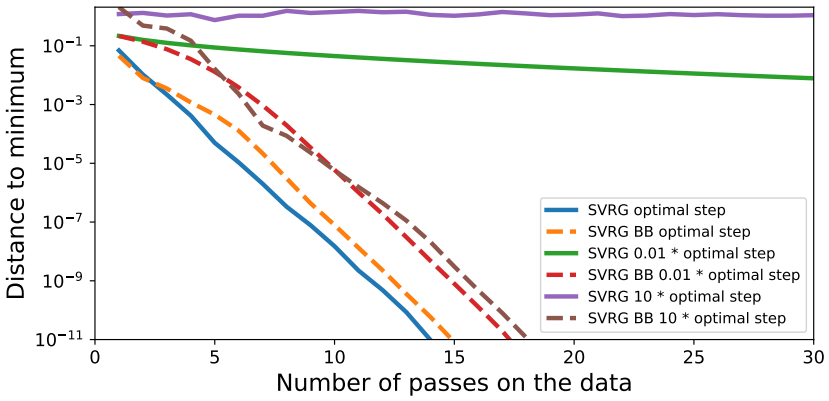

Figure V.9: Comparison of adaptive SVRG with traditional SVRG. While SVRG needs a fine tuning of its step, SVRG-BB converges quickly even when the initial step is not good.

6.4 Lower precision to accelerate algorithms

As presented in Section 5, `tick` relies on a C++ implementation. This enables vectorization that are processor instructions used to perform in parallel identical operations on a contiguous block of data [Dei12]. In this example we compare the computation time of two learners, the first one working with 32-bit single-precision floating-point numbers and the second with 64-bit double-precision. As more operations are run in parallel when single precision is used, the 32-bit learner performs quicker iterations than the 64-bit learner as shown in Figure V.10. However, as it works with lower precisions, it stops converging earlier.

```
from tick.dataset import fetch_tick_dataset
from tick.linear_model import LogisticRegression
from tick.plot import plot_history

X, y = fetch_tick_dataset('binary/adult/adult.trn.bz2')

learner_64 = LogisticRegression(solver='svrg')
learner_64.fit(X, y)

X_32, y_32 = X.astype('float32'), y.astype('float32')
learner_32 = LogisticRegression(solver='svrg')
learner_32.fit(X_32, y_32)

fig, axes = plt.subplots(1, 2, figsize=(8, 4), sharey=True)
plot_history([learner_32, learner_64], x='n_iter',
             labels=['float 32', 'float 64'], dist_min=True,
             log_scale=True, ax=axes[0])
plot_history([learner_32, learner_64], x='time',
             labels=['float 32', 'float 64'], dist_min=True,
             log_scale=True, ax=axes[1])
```

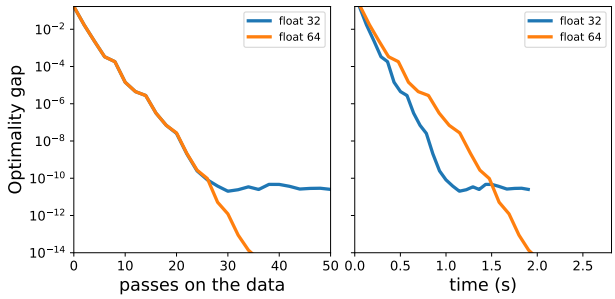

Figure V.10: Comparison of two identical learners working with different precisions. With lower precision, the iterations are quicker but the obtained result is not as precise.

7 Hawkes with non constant exogenous intensity

In the standard Hawkes model, we usually consider that the baseline μ, that is the exogenous intensity, is constant. However, this hypothesis is wrong in many cases. Let us take two examples: a dataset obtained by recording Twitter activity and one counting the orders of a finance marketplace. In both cases, the exogenous intensity

is not expected to be constant: a user is more likely to tweet during the day rather than in the middle of the night and exogenous events such as the opening of another finance marketplace can increase the number of orders. To tackle this issue, we focus on sum of exponentials kernels (see Section 2.4 of Chapter I for notations) which is the most scalable and the most commonly used parametrization. In this setting, the conditional intensity of node $i = 1, \dots, D$ with non constant baselines writes

$$\lambda^i(t|\, \mathscr{F}_t) = \mu^i(t) + \sum_{j=1}^{D} \sum_{u=1}^{U} \sum_{l:t_l^j < t} \alpha_u^{ij} \varphi_u\big(t - t_l^j\big)$$

where $\mu^i : \mathbb{R}^+ \to \mathbb{R}$ is the non constant exogenous intensity and $\varphi_u : t \mapsto \beta_u \exp(-\beta_u(t))$ for $u = 1, \dots, U$ is the kernel family. We parametrize the functions μ^i such that they are piecewise constant on P given subsets of $\mathbb{R}^+$, I_p for $p = 1, \dots, P$. Hence, for $i = 1, \dots, D$, the baselines write

$$\mu^i(t) = \sum_{p=1}^{P} \mu_p^i \mathbb{1}_{t \in I_p}.$$

Also, we suppose that these subsets do not intersect, $\bigcap_{p=1}^{P} I_p = \varnothing$, and that each subset I_p is a finite union of Q_p intervals, which all write

$$\forall\, p \in \{1, \dots, P\}, \quad \exists \{z_p^{l,1}, z_p^{u,1} \dots, z_p^{l,Q_p}, z_p^{u,Q_p}\}, \quad I_p = \bigcup_{q=1}^{Q_p} [z_p^{l,q}, z_p^{u,q})$$

where for each p the set of $\{z_p^{l,1}, z_p^{u,1}, z_p^{l,2}, z_p^{u,2}, \dots, z_p^{l,Q_p}, z_p^{u,Q_p}\}$ is sorted.

Then, we minimize the least square error to find the intensity with this parametrization that fits the best our set of observations. We choose least squares error because it is more scalable with the number of events and the loss is easier to optimize (see Chapters I, II and IV) but a similar analysis could be done with log-likelihood. With piecewise constant baselines and sum of exponentials kernels, the least squares error for node $i = 1, \dots, D$ from Equation (I.2) writes

$$\begin{aligned} R^i(\mu, \alpha, \mathscr{F}_T) = \int_0^T \mu^i(t)^2 \,\mathrm{d}t + 2 \int_0^T \Big(\mu^i(t) \sum_{j=1}^{D} \sum_{u=1}^{U} \alpha_u^{ij} g_u^j(t)\Big) \mathrm{d}t \\ + \int_0^T \Big(\sum_{j=1}^{D} \sum_{u=1}^{U} \alpha_u^{ij} g_u^j(t)\Big)^2 \mathrm{d}t - 2 \sum_{k=1}^{N_T} \mu^i(t_k^i) - 2 \sum_{k=1}^{N_T} \sum_{j=1}^{D} \sum_{u=1}^{U} \alpha_u^{ij} g_u^j(t_k^i) \end{aligned}$$

where $g_u^j(t) = \sum_{l:t_l^j < t} \beta_u \exp\big(-\beta_u(t - t_l^j)\big)$ for $j = 1, \dots, D$ and $u = 1, \dots, U$. As the subsets I_p do not intercept, the first term writes easily

$$\int_0^T \mu_i^2(t) \,\mathrm{d}t = \sum_{i=1}^{p} {\mu_p^i}^2 L_p \quad \text{where} \quad L_p = \sum_{q=1}^{Q_p} (z_p^{u,q} - z_p^{l,q})$$

is the sum of the lengths of the intervals whose union is I_p. To compute the second term, we compute p integrals on which the baselines μ^i is constant:

$$2\int_0^T \left(\mu^i(t)\sum_{j=1}^{D}\sum_{u=1}^{U}\alpha_u^{ij}g_u^j(t)\right)\mathrm{d}t = 2\sum_{j=1}^{D}\sum_{u=1}^{U}\alpha_u^{ij}\sum_{p=1}^{P}\mu_p^i\int_{[0,T]\cap I_p} g_u^j(t)\,\mathrm{d}t$$

where each integral can be evaluated as following:

$$M_{p,u}^j = \int_{[0,T]\cap I_p} g_u^j(t)\,\mathrm{d}t = \sum_{q=0}^{Q_p}\sum_{l:t_l^j<z_p^{u,q}} \left(e^{-\beta_u \max(0,z_p^{l,q}-t_l^j)} - e^{-\beta_u(z_p^{u,q}-t_l^j)}\right)$$

for $j = 1,\dots,D$, $p = 1,\dots,P$, and $u = 1,\dots,U$. The third and fifth terms remain unchanged whether the baselines are constant or not. Their computations are detailed in Proposition 1 of Chapter I. Finally, the fourth term writes

$$-2\sum_{k=1}^{N_T}\mu^i(t) = -2\sum_{p=1}^{P}\mu_p^i K_p^i \quad \text{where} \quad K_p^i = \sum_{t_k^i}\mathbb{1}_{t_k^i\in I_p}$$

and counts the number of events from node i occurring within I_p. With all these precomputed weights the least squares loss for node $i = 1,\dots,D$ writes

$$\begin{aligned}R^i(\mu,\alpha,\mathscr{F}_T) = {} & \sum_{i=1}^{p}{\mu_p^i}^2 L_p + 2\sum_{j=1}^{D}\sum_{u=1}^{U}\alpha_u^{ij}\sum_{p=1}^{P}\mu_p^i M_{p,u}^j + \sum_{j=1}^{D}\sum_{u=1}^{U}\sum_{u'=1}^{U}\alpha_u^{ij}\alpha_{u'}^{ij}K_{uu'}^{ij} \\ & + 2\sum_{j=1}^{D}\sum_{j'=1}^{D}\sum_{u=1}^{U}\sum_{u'=1}^{U}\alpha_u^{ij}\alpha_{u'}^{ij'}\Lambda_{uu'}^{ijj'} - 2\sum_{p=1}^{P}\mu_p^i K_p^i - 2\sum_{j=1}^{D}\sum_{u=1}^{U}\alpha_u^{ij}\sum_{k=1}^{N_T^i} g_u^j(t_k^i).\end{aligned}$$

This is a quadratic function in μ and α and thus is easy to optimize. With the new weights M, the original complexity for the weights computation in $\mathcal{O}(D^2U^2N_T)$ is increased by a new term in $\mathcal{O}(DUQN_T)$, where Q counts the number of intervals $[z^l, z^u)$ mapping $[0,T]$. The computation complexity of the error in $\mathcal{O}(D^3U^2)$ is simply increased by $\mathcal{O}(D^2UP)$ which is negligible in most cases.

In the following Python code, we simulate a Hawkes process parametrized with sum of exponentials kernels and a time varying baseline with a known period of length $\ell = 300$. The period in then divided in six intervals of equal length on which the baseline is constant. Namely, for $p = 1,\dots,P$ the subsets I_p write

$$I_p = [0,T]\cap\bigcup_{q=1}^{\lceil T/\ell\rceil+1}\left[((p-1)/p+q-1)\ell, (p/P+q-1)\ell\right)$$

For social network, this period would be set to 24 hours while in finance time series a period lasts as long as the market opening. In such cases, this setting infers

the prior we have on the seasonality of the exogenous intensity. In the proposed experiment the data is simulated. This allows us to compare the estimated intensity components with the generating parameters. Figure V.11 shows how tightly the kernels φ and the exogenous intensities are retrieved with only 6,000 events.

```
from tick.plot import plot_hawkes_baseline_and_kernels
from tick.hawkes import SimuHawkesSumExpKernels, HawkesSumExpKern

period_length = 300
baselines = [[.3, .5, .6, .4, .2, 0.],
             [.8, .5, .2, .3, .3, .4]]
decays = [.5, 2., 6.]
adjacency = [[[0., .1, .4], [.2, 0., .2]],
             [[0., 0., 0.], [.6, .3, 0.]]]

hawkes = SimuHawkesSumExpKernels(baseline=baselines, end_time=end_time,
                                 period_length=period_length,
                                 decays=decays, adjacency=adjacency)
hawkes.adjust_spectral_radius(0.5)
hawkes.simulate()

learner = HawkesSumExpKern(decays=decays,
                           n_baselines=len(baselines[0]),
                           period_length=period_length)

learner.fit(multi.timestamps)

plot_hawkes_baseline_and_kernels(learner, hawkes=hawkes)
```

8 Asynchronous stochastic solvers

This section is a short review on the recently introduced asynchronous stochastic solvers. These solvers benefit from parallel computations to speedup the computations in a single node multi-core setting in the context of composite sum minimization (see Chapter III), that writes as following

$$\min_{w \in \mathbb{R}^d} F(w) \quad \text{with} \quad F(w) = f(w) + g(w), \quad f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w),$$

where each convex functions f_i typically corresponds to a loss associated to the i-th observation of a dataset of n samples and the convex function g is a penalization term. To solve this problem, we can consider both batch solvers (see Section 2 of

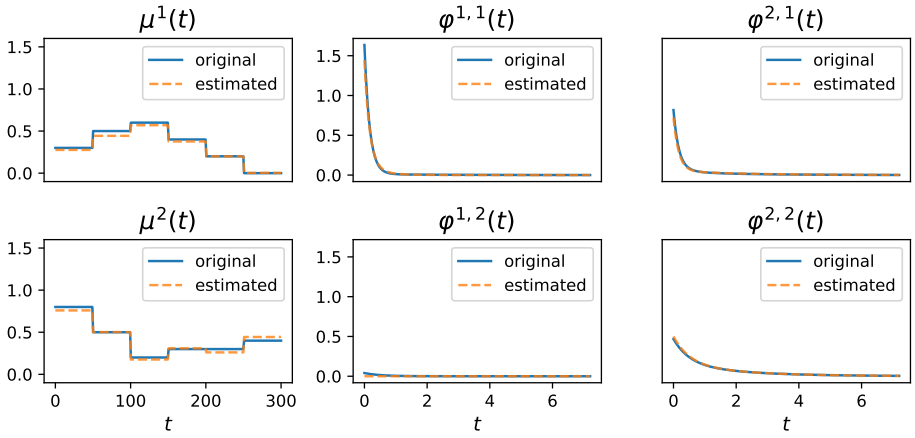

Figure V.11: Simulation and estimation of a Hawkes process with varying exogenous intensity μ on period of length $t = 300$. This example should how tightly the kernels φ and the exogenous intensities are retrieved with only 6,000 events

Chapter III) that process all observations at each iteration and stochastic solvers (see Section 3 of Chapter III) that perform an update each time they process an observation. In a big data context, when the number of observations n becomes large, the cost of batch methods might become prohibitive and stochastic methods benefit from their quick iterations. However, while batch algorithms are straightforward to parallelize, it is more complicated with stochastic algorithms. In a batch algorithm, one compute a full gradient over n observations and then perform a descent step with it. The bottleneck is the full gradient computation that can be split over several threads whereas the – comparatively cheap – descent step is done after synchronization. Conversely, stochastic algorithms have no such bottleneck and an immediate parallel implementation would lead to synchronize each time an observation is processed. Such an implementation is inefficient due to performance-destroying memory locks. To address this issue, lock-free algorithms have been introduced. such as Hogwild! [RRWN11], Kromagnon [RHS+15, MPP+17], PASSCoDe [HYD15] and ASAGA [LPLJ17, PLLJ17] that are respectively variants of SGD [RM51], SVRG [JZ13], SDCA [SSZ13] and SAGA [DBLJ14]. These algorithms rely on the assumption that we are working with sufficiently sparse datasets: the support of the observation i, $\mathrm{Supp}(x_i)$, has, on average, a cardinal several orders of magnitude smaller than the number of features d. In such a context, the updates might only modify a small part of the parameter variable w at each step and the collisions between threads are expected to be sufficiently rare to not disturb the overall behavior.

The aim of this section is to provide intuition and implementation details of lock-free algorithms that are implemented in `tick`. We do not provide the theoretical

guarantees derived for these algorithms but rather tips to implement them in an efficient manner. In this section we consider that the penalization g is separable meaning that $g : \mathbb{R}^d \to \mathbb{R}$ rewrites $g(w) = \sum_{j=1}^{d} g_j(w_j)$ where $g_j : \mathbb{R} \to \mathbb{R}$ for $j = 1, \dots, d$. This characteristic is verified by the most commonly used penalization techniques such as ℓ_1, ℓ_2 and elastic-net penalizations. Also, the implementations of ASAGA and PASSCode are given for generalized linear models, namely when the loss writes $f_i(w) = \varphi_i(w^\top x_i)$ where $\varphi_i : \mathbb{R} \to \mathbb{R}$ and $x_i \in \mathbb{R}^d$ is the i-th observation of the dataset. This family of models includes linear regression and logistic regression among others. While not strictly necessary, this allows easier and faster implementations for these algorithms.

Hogwild! Hogwild! implementation is detailed in Algorithm V.2, it simply consists in executing SGD updates on a shared vector $w \in \mathbb{R}^d$ from several parallel threads. The only refinement is that the iterate updates are done atomically, meaning that at Line 5, the value of w_j cannot be modified by an external thread between the read and write operations on w_j. Beyond vanilla SGD, more sophisticated stochastic gradient descent algorithms with variance reduction also have their own asynchronous variant.

Algorithm V.2 Hogwild! (Asynchronous SGD)

Require: Starting point w, a step size η

1: **loop**
2: Sample at random i in $\{1, \dots, n\}$
3: $\Delta w \leftarrow -\eta \nabla f_i(w)$ ▷ inconsistent read of w
4: **for** j in $\mathrm{Supp}(\Delta w)$ **do**
5: $w_j \leftarrow w_j + \Delta w_j$ ▷ atomic update
6: **end for**
7: **end loop**

Kromagnon First, Kromagnon (asynchronous SVRG) works as Hogwild! and performs atomic SVRG updates in parallel on a shared vector w. However, SVRG updates use a dense vector, $\Delta w_{\text{dense}} = -\eta\big(\nabla f_i(w) - \nabla f_i(\widetilde{w}) + \nabla f(\widetilde{w})\big)$, to apply the variance reduction technique. This dense operation might be transformed into a strictly equivalent sparse operation with the lazy updates [PLT+16]. But, the lazy updates are not adapted to a parallel setting because they would involve a lot of synchronizations. Hence, [MPP+17] has introduced a sparse formulation that is equivalent to the dense case *in expectation*. The dense part of Δw, $\nabla f(\widetilde{w})$, is multiplied by the $d \times d$ diagonal matrix D_i that is defined as $\mathrm{diag}(D_i)_j = p_j^{-1} \mathbb{1}_{j \in \mathrm{Supp}(\nabla f_i(w))}$ where p_j is the probability that feature the j belongs to $\mathrm{Supp}(\nabla f_i(w))$ when i is sampled

at random in $\{1,\dots,n\}$. This normalization ensures that $\mathbb{E}[D_i \nabla f(\widetilde{w})] = \nabla f(\widetilde{w})$ where $D_i \nabla f(\widetilde{w})$ has the same support as $\nabla f_i(w)$. Kromagnon's implementation is given in Algorithm V.3. The same technique can be used to deal with the composite case when the penalization is separable. Hence, we rather use $g_{D_i} : w \mapsto D_i g(w)$ that is an unbiased estimator of $g(w)$, $\mathbb{E}[D_i g(w)] = g(w)$, such that $g_{D_i}(w)$ has the same support as $\nabla f_i(w)$. In practice we use the corresponding proximal operator that writes for all $j = 1,\dots,d$: $[\mathrm{prox}_{\eta g_{D_i}}(w)]_j = \mathrm{prox}_{\eta/p_j\, g_j}(x)\mathbb{1}_{j \in \mathrm{Supp}(\nabla f_i(w))}$. While the original paper does not mention this extension to the composite case, this procedure that was introduced for ASAGA in [PLLJ17] works in practice when applied to Kromagnon.

Algorithm V.3 Kromagnon (Asynchronous SVRG)

Require: Starting point w, a step size $\eta > 0$

1: **for** $t = 0, 1, \dots$ **do**
2: $\quad \widetilde{w} \leftarrow w$
3: $\quad \widetilde{\mu} \leftarrow \nabla f(\widetilde{w})$
4: $\quad$ **while** less than m indices have been sampled **do**
5: $\quad\quad$ **keep doing in parallel**
6: $\quad\quad\quad$ Sample at random i in $\{1,\dots,n\}$
7: $\quad\quad\quad \Delta w \leftarrow -\eta\big(\nabla f_i(w) - \nabla f_i(\widetilde{w}) + D_i\widetilde{\mu}\big)$ ▷ inconsistent read of w
8: $\quad\quad\quad$ **for** j in $\mathrm{Supp}(\Delta w)$ **do**
9: $\quad\quad\quad\quad \Delta w_j \leftarrow \mathrm{prox}_{\eta/p_j\, g_j}\big(w_j + \Delta w_j\big) - w_j$
10: $\quad\quad\quad\quad w_j \leftarrow w_j + \Delta w_j$ ▷ atomic update
11: $\quad\quad\quad$ **end for**
12: $\quad\quad$ **end parallel loop**
13: $\quad$ **end while**
14: **end for**

ASAGA Then ASAGA and Prox-ASAGA are the parallel implementations of SAGA algorithm. It uses the same strategy as Kromagnon to turn the dense update of SAGA into a sparse one that is equivalent in expectation. Also, a second atomic operation is used to keep the relationship $\Psi = \sum_{i=1}^{n} \psi_i x_i$ valid at all time. This is of great importance as this ensures that $\mathbb{E}[\psi_i x_i] = \Psi$ is maintained across iterations. The implementation provided in Algorithm V.4 is specialized for generalized linear model where $\nabla f_i(w)$ writes $\varphi_i'(w^\top x_i)x_i$. In the generic case, we would store vectors $\psi_{\text{generic},i} \in \mathbb{R}^d$ and atomic update of Line 6 of the algorithm would become much more expensive.

PASSCoDe PASSCoDe is the parallel version of SDCA. For a clearer implementation, besides the specification for generalized linear models, we also suppose that

Algorithm V.4 ASAGA (Asynchronous SAGA)

Require: Starting point w, a step size $\eta > 0$, $\psi_i \in \mathbb{R}$ for $i = 1, \dots, n$

$\Psi \leftarrow \sum_{i=1}^n \psi_i x_i$
keep doing in parallel
  Sample at random i in $\{1, \dots, n\}$
  $\Delta\psi = \varphi_i'(w^\top x_i) - \psi_i$ ▷ inconsistent read of w
  $\Delta w = -\eta\big(\Delta\psi\, x_i + \frac{1}{n} D_i \Psi\big)$
  $\psi_i \leftarrow \psi_i + \Delta\psi$ ▷ atomic update
  for j in $\mathrm{Supp}(\Delta w)$ **do**
    $\Psi_j \leftarrow \Psi_j + \frac{1}{n}\Delta\psi x_{ij}$ ▷ atomic update
    $\Delta w_j \leftarrow \mathrm{prox}_{\eta/p_j g_j}\big(w_j + \Delta w_j\big) - w_j$
    $w_j \leftarrow w_j + \Delta w_j$ ▷ atomic update
  end for
end parallel loop

the penalization (that is strongly convex by assumption) can be decomposed into $g(w) = \frac{\lambda}{2}\|w\|^2 + h(w)$ where h is a convex, separable and prox-capable function such as $w \mapsto \|w\|_1$ for elastic-net penalization or $w \mapsto 0$ for ℓ_2 penalization. Like ASAGA maintains $\Psi = \sum_{i=1}^n \psi_i x_i$, PASSCoDe must maintain the following relation across iterations $v = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i$ thanks to atomic operations. This relation is indeed at the heart of the variance reduction technique (see Section *SDCA as Variance Reduction* in [JZ13]). Unlike SVRG and SAGA, the updates are sparse by design, since finding

$$\Delta\alpha_i = \operatorname*{argmax}_{\Delta\alpha} \tfrac{1}{n}\varphi_i^*\big(-\alpha - \Delta\alpha\big) - \tfrac{\lambda}{2}\big\|w + \tfrac{1}{\lambda n}\Delta\alpha_i x_i\big\|^2,$$

where φ_i^* is the Fenchel convex conjugate of φ_i, is strictly equivalent to find

$$\Delta\alpha_i = \operatorname*{argmax}_{\Delta\alpha} \varphi_i^*\big(-\alpha_i - \Delta\alpha\big) - \tfrac{1}{2\lambda n}\Delta\alpha^2\|x_i\|^2 - \Delta\alpha\, w^\top x_i,$$

where the inner product $w^\top x_i = \sum_{j\in\mathrm{Supp}(x_i)} \mathrm{prox}_{\frac{1}{\lambda}h_j}(v_j) x_{ij}$ is a sparse operation that lazily evaluate the entries of w. This leads to the implementation detailed in Algorithm V.5.

About atomic operations Atomic operations are at the heart of these algorithms. First, for ASAGA and PASSCoDe, they ensure that the variance reduction relation is maintained. Second, they also impact how the iterate w is updated. All algorithms perform inconsistent read of w to evaluate the descent direction Δw, but, once this direction is determined, the atomic update of the iterate w ensures that it is fully applied. We numerically investigate on the necessity of these atomic

Algorithm V.5 PASSCoDe (Asynchronous SDCA)

Require: Starting dual $\alpha \in \mathbb{R}^n$

1: $v \leftarrow \frac{1}{\lambda n} \sum_{i=1}^n \alpha_i x_i$
2: **keep doing in parallel**
3: Sample at random i in $\{1, \dots, n\}$
4: Find $\Delta\alpha_i \in \mathbb{R}$ that maximizes
5: $\varphi_i^*\big(-\alpha_i - \Delta\alpha\big) - \frac{1}{2\lambda n}\Delta\alpha^2 \|x_i\|^2 - \Delta\alpha\, w^\top x_i$ ▷ inconsistent read of w
6: **for** j in Supp(x_i) **do**
7: $v_j \leftarrow v_j + \frac{1}{\lambda n}\Delta\alpha_i x_{ij}$ ▷ atomic update
8: **end for**
9: $\alpha_i \leftarrow \alpha_i + \Delta\alpha_i$ ▷ atomic update
10: **end parallel loop**
11: $w \leftarrow \text{prox}_{\frac{1}{\lambda}h}(v)$

operations in the following paragraph. Also, all our atomic operations are implemented in C++ using `std::memory_order_relax` orderings instead of the default `std::memory_order_seq_cst`. With this ordering, the order of the atomic operations is not guaranteed but this is harmless in our situation since we do not need such a synchronization but only must ensure that all increments will eventually be done. Typically, `std::memory_order_relax` is used to increment counters, such as the reference counters of `std::shared_ptr` [7] that meets the same requirements.

Experiments To evaluate the importance of the atomic operations we test the previous algorithms with and without their atomic operations. We call *atomic iterate*, the atomic update of the iterate w, namely Line 10 of Algorithm V.3 (Kromagnon) and Line 10 of Algorithm V.4 (ASAGA) and *atomic variance reduction*, the atomic operations that maintain the variance reduction relationship, namely Lines 6 and 8 of Algorithm V.4 (ASAGA) and Lines 7 and 9 of Algorithm V.5 (PASSCoDe). The labels given to the solvers built from each of these variants are provided in Table V.3. We assess the performances and the speedup of these algorithms on a machine with 2 Intel Xeon E5-1660v4 processors with 8 3.2GHz cores each, that we use to minimize an $\ell_1 + \ell_2$-regularized logistic loss on the RCV1 dataset [LYRL04]. This penalization has been chosen to be both non-smooth (thanks to its ℓ_1 part) and compatible with SDCA (thanks to its ℓ_2 part). This dataset has $n = 804,401$ rows and $d = 47,236$ features that are 0.16% sparse. In Figure V.12, we display the convergence plots of the algorithms presented in Table V.3, that is to say the time needed to reach a certain precision level on the objective function $F(w)$. For each algorithm we have

[7] This example comes from the official documentation: https://en.cppreference.com/w/cpp/atomic/memory_order#Relaxed_ordering.

Table V.3: Label given to asynchronous solvers depending on which operations are atomic. Symbol ✓ means that the atomic operation is kept, ✗ means it has been discarded and – that it does not apply for this solver.

	Atomic iterate	Atomic variance reduction
SVRG	✗	–
Kromagnon	✓	–
SAGA	✗	✗
ASAGA$_w$	✓	✗
ASAGA$_\Psi$	✗	✓
ASAGA	✓	✓
SDCA	–	✗
PASSCoDe	–	✓

run the experiment with 1 core (classic sequential setting), 4 cores (laptop setting) and 16 cores (powerful computing machine). We observe that the algorithms that do not maintain the variance reduction relation across iterations, SAGA, ASAGA$_w$ and SDCA, have poor convergence properties when multi-threaded. Indeed, they lose the key property of variance reduced stochastic solvers, that is having the descent direction computed in w tending to 0 when w approaches w^*. On the contrary, solvers that only lack of atomic iterate updates, SVRG and ASAGA$_\Psi$, have good performances. SVRG is faster than Kromagnon in a 4 cores setting and similar to Kromagnon with 16 cores while ASAGA$_\Psi$ is better than ASAGA in both settings. To have a better understanding of the behaviors of these algorithms we examine the time and the number of passes on the data needed to obtain a 10^{-8} precision level. In Figure V.13, we display the time speedup of the five algorithms that reach the minimum. To quantify the gain of running the algorithm in parallel, we call *time speedup* the ratio $\frac{\text{time needed by sequential algorithm}}{\text{time needed by parallel algorithm}}$ whose theoretical optimal value is the number of cores involved. Figure V.13 shows how faster the parallel algorithms reach a 10^{-8} objective precision on the RCV1 dataset. For example, ASAGA and ASAGA$_\Psi$ algorithms are five to six times faster for this task when run in parallel on 16 cores than the sequential version. However, algorithms such as SVRG and PASSCoDe do not scale as well. In Figure V.14, we report the number of epochs (the total number of times the data is seen) needed by each algorithm to reach 10^{-8} precision. While, the number of epochs needed by ASAGA, ASAGA$_\Psi$ and Kromagnon seems constant, for SVRG and PASSCoDe, this number grows with the number of threads. This is probably one reason that makes these two algorithms have less good speedups on this task. We cannot conclude which version of each algorithm is the best based on

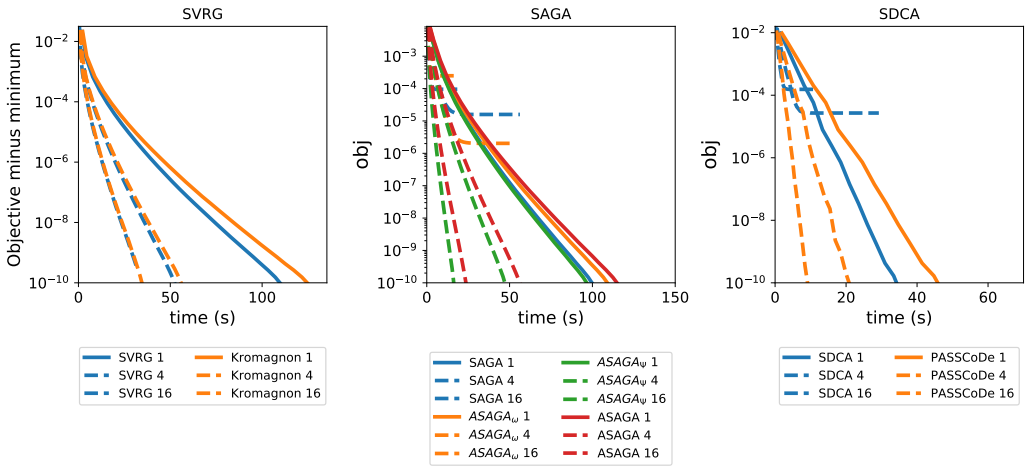

Figure V.12: Convergence plots of the algorithms presented in Table V.3 on the RCV1 dataset. The number next to the algorithm's label denotes the number of cores that were used during the training. It appears that solvers that are missing atomic variance reduction (namely SAGA, ASAGA$_w$ and SDCA) do not have linear convergence in a multi-threaded context. However, solvers that only lack of atomic iterate updates (SVRG and ASAGA$_\Psi$) keep their linear speed and are sometimes faster than their full atomic counterpart (resp. Kromagnon and ASAGA).

this sole example. Still, we can make two conclusions. First, if no atomic operations guarantee the variance reduction relation (see SAGA, ASAGA$_w$ and SDCA), the convergence properties are highly impacted. Second, using atomic operations to update the iterate might lead to a better speedup (see Kromagnon vs. SVRG) but also slows the algorithm down. In our example, this makes us prefer the non atomic iterate variants (SVRG and ASAGA$_\Psi$) over the full atomic one (Kromagnon and ASAGA) presented in the original articles.

Acknowledgments

The authors thank the Data-science Initiative of École polytechnique and Intel® for supporting `tick` development.

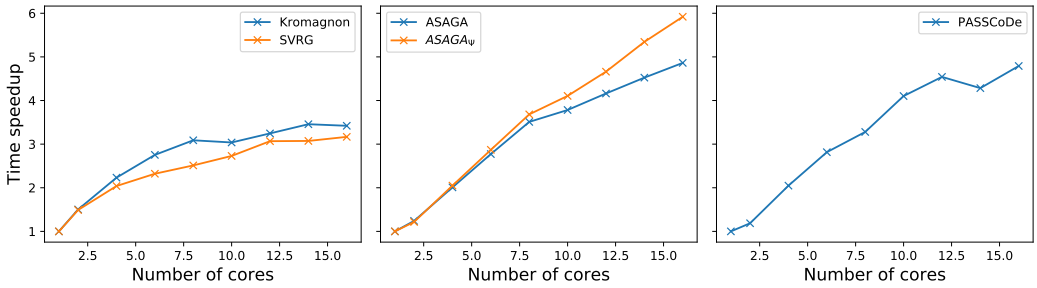

Figure V.13: Time speedup achieved by the algorithms presented in Table V.3 to reach 10^{-8} precision on the RCV1 dataset. The solvers that are missing atomic variance reduction (namely SAGA, ASAGA$_w$ and SDCA) are not represented since they do not reach a 10^{-8} precision in a reasonable time. We observe that on this example, ASAGA$_\Psi$ and ASAGA have the best time speedup.

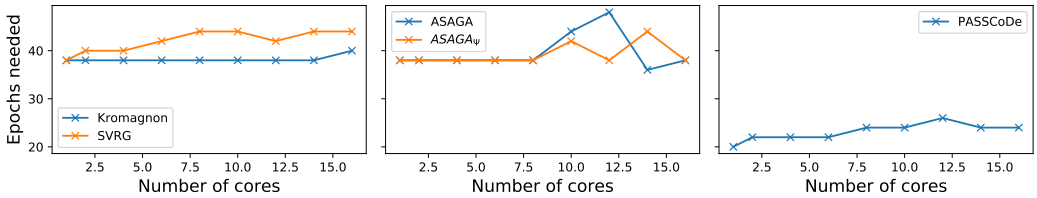

Figure V.14: Number of epochs needed to reach 10^{-8} precision on the RCV1 dataset.

Bibliography

[ABG+17] M. Achab, E. Bacry, S. Gaïffas, I. Mastromatteo, and J.-F. Muzy. Uncovering causality from multivariate hawkes integrated cumulants. In *International Conference on Machine Learning*, pages 1–10, 2017.

[ABGK12] P. K. Andersen, O. Borgan, R. D. Gill, and N. Keiding. *Statistical models based on counting processes*. Springer Science & Business Media, 2012.

[Bac10] F. Bach. Self-concordant analysis for logistic regression. *Electronic Journal of Statistics*, 4:384–414, 2010.

[BB88] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

[BB16] H. H. Bauschke and M. Bolte, J. Teboulle. A descent lemma beyond lipschitz gradient continuity: first-order methods revisited and applications. *Mathematics of Operations Research*, 42(2):330–348, 2016.

[BBDV09] M. Bertero, P. Boccacci, G. Desiderà, and G. Vicidomini. Image deblurring with poisson data: from cells to galaxies. *Inverse Problems*, 25(12):123006, 2009.

[BBH12] C. Blundell, J. Beck, and K. A. Heller. Modelling reciprocating relationships with hawkes processes. In *Advances in Neural Information Processing Systems*, pages 2600–2608, 2012.

[BDHM13] E. Bacry, S. Delattre, M. Hoffmann, and J.-F. Muzy. Modelling microstructure noise with mutually exciting point processes. *Quantitative Finance*, 13(1):65–77, 2013.

[Ber99] D. P. Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[BF10] H. C. Boshuizen and E. Feskens. Fitting additive poisson models. *Epidemiologic Perspectives & Innovations*, 7(1):4, 2010.

[BFK+96] D. Beazley, W. Fulton, M. Köppe, L. Johnson, and R. Palmer. Swig: Simplified wrapper and interface generator. *University of Utah, Salt Lake City, Utah*, 84112, 1996.

[BGM18] E. Bacry, S. Gaïffas, and J.-F. Muzy. Concentration inequalities for matrix martingales in continuous time. *Probability Theory and Related Fields*, 170:525, 2018.

[BJM16] E. Bacry, T. Jaisson, and J.-F. Muzy. Estimation of slowly decreasing hawkes kernels: application to high-frequency order book dynamics. *Quantitative Finance*, 16(8):1179–1201, 2016.

[BL07] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

[BLB+13] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[BM06] P. L. Bartlett and S. Mendelson. Empirical minimization. *Probability Theory and Related Fields*, 135(3):311–334, 2006.

[BM14] E. Bacry and J.-F. Muzy. Second order statistics characterization of hawkes processes and non-parametric estimation. *arXiv preprint arXiv:1401.0903*, 2014.

[BMM15] E. Bacry, I. Mastromatteo, and J.-F. Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(01):1550005, 2015.

[BRT09] P. J. Bickel, Y. Ritov, and A. B. Tsybakov. Simultaneous analysis of lasso and dantzig selector. *The Annals of Statistics*, 37(4):1705–1732, 2009.

[BT09] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[Bur03] W. J. Burroughs. *Weather cycles: real or imaginary?* Cambridge University Press, 2003.

[BV04] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Cau47] A. Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[CCA+09] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[CP11] P. L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011.

[CPC09] Y. Chen, D. Pavlov, and J. F. Canny. Large-scale behavioral targeting. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 209–218. ACM, 2009.

[CS08] R. Crane and D. Sornette. Robust dynamic classes revealed by measuring the response function of a social system. *Proceedings of the National Academy of Sciences*, 105(41):15649–15653, 2008.

[CT04] E. J. Candès and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 12(51):4203–4215, 2004.

[CT10] E. J. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

[DBLJ14] A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[DBS13] C. DuBois, C. Butts, and P. Smyth. Stochastic blockmodeling of relational event dynamics. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pages 238–246, 2013.

[Dei12] M. Deilmann. A guide to vectorization with intel c++ compilers. *Intel Corporation, April*, 2012.

[DHS11] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[dMB04] M. A. de Menezes and A.-L. Barabási. Fluctuations in network dynamics. *Phys. Rev. Lett.*, 92:028701, Jan 2004.

[DRSS14] N. Daneshmand, M. Rodriguez, L. Song, and B. Schölkpof. Estimating diffusion network structure: Recovery conditions, sample complexity, and a soft-thresholding algorithm. *International Conference on Machine Learning*, 2014.

[DVG+16] A. De, I. Valera, N. Ganguly, S. Bhattacharya, and M. G. Rodriguez. Learning and forecasting opinion dynamics in social networks. In *Advances in Neural Information Processing Systems*, pages 397–405, 2016.

[DVJ07] D. J. Daley and D. Vere-Jones. *An introduction to the theory of point processes: volume II: general theory and structure.* Springer Science & Business Media, 2007.

[ELL11] P. Embrechts, T. Liniger, and L. Lin. Multivariate hawkes processes: an application to financial data. *Journal of Applied Probability*, 48(A):367–378, 2011.

[FS15] V. Filimonov and D. Sornette. Apparent criticality and calibration issues in the hawkes self-excited point process model: application to high-frequency financial data. *Quantitative Finance*, 15(8):1293–1314, 2015.

[FVC15] K. Fernandes, P. Vinagre, and P. Cortez. A proactive intelligent decision support system for predicting the popularity of online news. In *Portuguese Conference on Artificial Intelligence*, pages 535–546. Springer, 2015.

[FWR+15] M. Farajtabar, Y. Wang, M. G. Rodriguez, S. Li, H. Zha, and L. Song. Coevolve: A joint point process model for information diffusion and network co-evolution. In *Advances in Neural Information Processing Systems*, pages 1954–1962, 2015.

[GL15] A. Galves and E. Löcherbach. Modeling networks of spiking neurons as interacting processes with memory of variable length. *arXiv preprint arXiv:1502.06446*, 2015.

[GRLS13] M. Gomez-Rodriguez, J. Leskovec, and B. Schölkopf. Modeling information propagation with survival theory. *International Conference on Machine Learning*, 2013.

[Ham94] J. D. Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[Haw71a] A. G. Hawkes. Point spectra of some mutually exciting point processes. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 438–443, 1971.

[Haw71b] A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[HBB13] S. J. Hardiman, N. Bercot, and J.-P. Bouchaud. Critical reflexivity in financial markets: a hawkes process analysis. *The European Physical Journal B*, 86(10):442, 2013.

[HL16] E. Hazan and H. Luo. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, pages 1263–1271, 2016.

[HMW12] Z. T. Harmany, R. F. Marcia, and R. M. Willett. This is spiral-tap: Sparse poisson intensity reconstruction algorithms—theory and practice. *IEEE Transactions on Image Processing*, 21(3):1084–1096, 2012.

[HO74] A. G. Hawkes and D. Oakes. A cluster process representation of a self-exciting process. *Journal of Applied Probability*, 11(3):493–503, 1974.

[HRBR15] N. R. Hansen, P. Reynaud-Bouret, and V. Rivoirard. Lasso and probabilistic inequalities for multivariate point processes. *Bernoulli*, 21(1):83–143, 2015.

[HYD15] C.-J. Hsieh, H.-F. Yu, and I. Dhillon. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. In *ICML*, volume 15, pages 2370–2379, 2015.

[ISG13] T. Iwata, A. Shah, and Z. Ghahramani. Discovering latent influence in online social activities via shared cascade poisson processes. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 266–274. ACM, 2013.

[JZ13] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[KB15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.

[KLT11] V. Koltchinskii, K. Lounici, and A. B. Tsybakov. Nuclear-norm penalization and optimal rates for noisy low-rank matrix completion. *The Annals of Statistics*, 39(5):2302–2329, 2011.

[Kol11] V. Koltchinskii. *Oracle Inequalities in Empirical Risk Minimization and Sparse Recovery Problems: Saint-Flour XXXVIII-2008*, volume 2033. Springer, 2011.

[LA14] S. Linderman and R. Adams. Discovering latent network structure in point process data. In *International Conference on Machine Learning*, pages 1413–1421, 2014.

[LBK09] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD*. ACM, 2009.

[Les08] J. Leskovec. *Dynamics of large networks*. PhD thesis, Machine Learning Department, Carnegie Mellon University, 2008.

[Lew95] A. S. Lewis. The convex analysis of unitarily invariant matrix functions. *Journal of Convex Analysis*, 2(1):173–183, 1995.

[LFN18] H. Lu, R. M. Freund, and Y. Nesterov. Relatively smooth convex optimization by first-order methods, and applications. *SIAM Journal on Optimization*, 28(1):333–354, 2018.

[Lic13] M. Lichman. UCI machine learning repository, 2013.

[LK04] H. Lütkepohl and M. Krätzig. *Applied time series econometrics*. Cambridge university press, 2004.

[LM11] E. Lewis and G. Mohler. A nonparametric em algorithm for multiscale hawkes processes. *Journal of Nonparametric Statistics*, 1(1):1–20, 2011.

[LMBB12] E. Lewis, G. Mohler, P. J. Brantingham, and A. L. Bertozzi. Self-exciting point process models of civilian deaths in iraq. *Security Journal*, 25(3):244–264, 2012.

[LPLJ17] R. Leblond, F. Pedregosa, and S. Lacoste-Julien. Asaga: asynchronous parallel saga. *In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[LSK17] R. Lemonnier, K. Scaman, and A. Kalogeratos. Multivariate hawkes processes for large-scale inference. In *AAAI*, pages 2168–2174, 2017.

[LSV+16] M. Lukasik, P. K. Srijith, D. Vu, K. Bontcheva, A. Zubiaga, and T. Cohn. Hawkes processes for continuous time sequence classification: an application to rumour stance classification in twitter. In *Proceedings of 54th Annual Meeting of the Association for Computational Linguistics*, pages 393–398. Association for Computational Linguistics, 2016.

[LV14] R. Lemonnier and N. Vayatis. Nonparametric markovian learning of triggering kernels for mutually exciting and mutually inhibiting multivariate hawkes processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 161–176. Springer, 2014.

[LYRL04] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

[Mas07] P. Massart. *Concentration inequalities and model selection*, volume 1896. Springer, 2007.

[Moh13] G. Mohler. Modeling and estimation of multi-source clustering in crime and security data. *The Annals of Applied Statistics*, 7(3):1525–1539, 2013.

[MPP+17] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

[MRC17] S. Moro, P. Rita, and J. Coelho. Stripping customers' feedback on hotels through data mining: the case of las vegas strip. *Tourism Management Perspectives*, 23:41–52, 2017.

[MRV16] S. Moro, P. Rita, and B. Vala. Predicting social media performance metrics and evaluation of the impact on brand building: A data mining approach. *Journal of Business Research*, 69(9):3341–3351, 2016.

[MSB+11] G. O. Mohler, M. B. Short, P. J. Brantingham, F. P. Schoenberg, and G. E. Tita. Self-exciting point process modeling of crime. *Journal of the American Statistical Association*, 2011.

[Nes83] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[Nes13] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[Noc80] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[NW06] J. Nocedal and S. J. Wright. *Nonlinear Equations*. Springer, 2006.

[Oga78] Y. Ogata. The asymptotic behaviour of maximum likelihood estimators for stationary point processes. *Annals of the Institute of Statistical Mathematics*, 30(1):243–261, 1978.

[Oga81] Y. Ogata. On lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, 27(1):23–31, 1981.

[Oga88] Y. Ogata. Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical association*, 83(401):9–27, 1988.

[Oga98] Y. Ogata. Space-time point-process models for earthquake occurrences. *Annals of the Institute of Statistical Mathematics*, 50(2):379–402, 1998.

[Oga99] Y. Ogata. Seismicity analysis through point-process modeling: A review. *Pure & Applied Geophysics*, 155(2-4):471, 1999.

[PLLJ17] F. Pedregosa, R. Leblond, and S. Lacoste-Julien. Breaking the nonsmooth barrier: A scalable parallel method for composite optimization. In *Advances in Neural Information Processing Systems*, pages 56–65, 2017.

[PLR+99] M. R. Pino, L. Landesa, J. L. Rodriguez, F. Obelleiro, and R. J. Burkholder. The generalized forward-backward method for analyzing the scattering from targets on ocean-like rough surfaces. *IEEE Transactions on Antennas and Propagation*, 47(6):961–969, 1999.

[PLT+16] X. Pan, M. Lam, S. Tu, D. Papailiopoulos, C. Zhang, M. I. Jordan, K. Ramchandran, and C. Ré. Cyclades: Conflict-free asynchronous machine learning. In *Advances in Neural Information Processing Systems*, pages 2568–2576, 2016.

[PTLJ18] R. L. Priol, A. Touati, and S. Lacoste-Julien. Adaptive stochastic dual coordinate ascent for conditional random fields. *arXiv preprint arXiv:1712.08577*, (291), 2018.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Qia99] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[QRTF16] Z. Qu, P. Richtárik, M. Takác, and O. Fercoq. Sdna: Stochastic dual newton ascent for empirical risk minimization. In *International Conference on Machine Learning*, pages 1823–1832, 2016.

[RBL17] M. Rambaldi, E. Bacry, and F. Lillo. The role of volume in order book dynamics: a multivariate hawkes process analysis. *Quantitative Finance*, 17(7):999–1020, 2017.

[RBR10] P. Reynaud-Bouret and V. Rivoirard. Near optimal thresholding estimation of a poisson intensity on the real line. *Electronic journal of statistics*, 4:172–238, 2010.

[RBS11] M. Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. *International Conference on Machine Learning*, 2011.

[RGV14] E. Richard, S. Gaïffas, and N. Vayatis. Link prediction in graphs with autoregressive features. *Journal of Machine Learning Research*, 2014.

[RHS+15] S. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655, 2015.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[RM51] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[RRS11] F. Ricci, L. Rokach, and B. Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[RRWN11] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[Sca82] J. D. Scargle. Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data. *The Astrophysical Journal*, 263:835–853, 1982.

[SLRB17] M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[SS16] S. Shalev-Shwartz. Sdca without duality, regularization, and individual convexity. In *International Conference on Machine Learning*, pages 747–754, 2016.

[SSZ13] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

[SSZ14] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *International Conference on Machine Learning*, pages 64–72, 2014.

[STD17] T. Sun and Q. Tran-Dinh. Generalized self-concordant functions: a recipe for newton-type methods. *Mathematical Programming*, pages 1–69, 2017.

[Tay08] S. J. Taylor. *Modelling financial time series*. world scientific, 2008.

[TDKC15] Q. Tran-Dinh, A. Kyrillidis, and V. Cevher. Composite self-concordant minimization. *The Journal of Machine Learning Research*, 16(1):371–416, 2015.

[TFSZ15] L. Tran, M. Farajtabar, L. Song, and H. Zha. Netcodec: Community detection from individual activities. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 91–99. SIAM, 2015.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[Tib11] R. Tibshirani. Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(3):273–282, 2011.

[TMDQ16] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian. Barzilai-borwein step size for stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 685–693, 2016.

[Tro12] J. A. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012.

[Tsa05] R. S. Tsay. *Analysis of financial time series*, volume 543. John Wiley & Sons, 2005.

[VDG00] S. Van De Geer. *Empirical Processes in M-estimation*, volume 105. Cambridge university press Cambridge, 2000.

[XFZ16] H. Xu, M. Farajtabar, and H. Zha. Learning granger causality for hawkes processes. In *International Conference on Machine Learning*, pages 1717–1726, 2016.

[XZ14] L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

[YZ13] S.-H. Yang and H. Zha. Mixture of mutually exciting processes for viral diffusion. In *International Conference on Machine Learning*, pages 1–9, 2013.

[Zei12] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[ZZ15] P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1–9, 2015.

[ZZS13a] K. Zhou, H. Zha, and L. Song. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *AISTATS*, volume 31, pages 641–649, 2013.

[ZZS13b] K. Zhou, H. Zha, and L. Song. Learning triggering kernels for multi-dimensional hawkes processes. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1301–1309, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

ANNEXE A

Résumé des contributions

Cette introduction est un bref résumé des chapitres de cette thèse. Elle donne une vision d'ensemble du travail présenté et passe parfois rapidement sur les détails techniques. Ces détails sont dûment étayés dans les chapitres correspondants.

1 Les processus de Hawkes

Les processus ponctuels temporels sont utilisés pour étudier les séries d'événements en temps continu. A l'inverse des séries temporelles, ils ne dépendent d'aucune résolution temporelle définie à l'avance et peuvent ainsi étudier plusieurs échelles de temps à la fois. Ce chapitre est une brève introduction sur les processus ponctuels temporels et en particulier sur les processus de Hawkes. D'avantage de détails peuvent être trouvés dans l'ouvrage de référence [DVJ07].

1.1 Processus ponctuels temporels

Nous associons à un ensemble de points distincts du temps $\{t_1, \dots, t_n\}$, ayant lieu dans un intervalle $[0, T]$, le processus de comptage $N_t = \sum_{t_k} \mathbb{1}_{t_k \leq t}$. Ce processus de comptage est un processus aléatoire dont la distribution est caractérisée par une *fonction d'intensité* $\lambda(t|\mathscr{F}_t)$ qui donne la probabilité infinitésimale avec laquelle un événement va se produire au temps t étant donnée l'information $\mathscr{F}_t$ disponible jusqu'au temps t exclu. Elle s'écrit

$$\lambda(t|\mathscr{F}_t) = \lim_{dt \to 0} \frac{\mathbb{P}(N_{t+dt} - N_t = 1 |\ \mathscr{F}_t)}{dt}.$$

Dans le cas le plus simple, cette intensité est constante et le processus associé est appelé processus de Poisson homogène. Il décrit un phénomène sans mémoire et avec une probabilité d'apparition d'un événement constante au cours du temps.

Qualité de l'ajustement. On appelle qualité de l'ajustement ou *goodness-of-fit* une fonction caractérisant à quel point un modèle statistique s'ajuste à un ensemble d'observations. La *goodness-of-fit* la plus utilisée est la fonction de vraisemblance donnée par [DVJ07]. Par commodité, nous considérons plutôt l'opposé de son logarithme en tant que mesure d'erreur. Il est alors donnée par

$$-\log L(\mathscr{F}_T) = \int_0^T \lambda(s|\mathscr{F}_s)\,\mathrm{d}s - \sum_{k=1}^{N_T} \log \lambda(t_k|\mathscr{F}_{t_k}).$$

Une autre mesure est l'erreur des moindres carrés inspirée par le principe de minimisation du risque empirique. Pour un processus ponctuel, elle s'écrit (voir [RBR10, HRBR15])

$$R(\mathscr{F}_T) = \int_0^T \lambda(s|\mathscr{F}_s)^2\,\mathrm{d}s - 2\sum_{k=1}^{N_T} \lambda(t_k|\mathscr{F}_{t_k}). \tag{A.1}$$

En supposant que N_t a une intensité réelle inconnue notée λ^*, $\mathbb{E}[R(\mathscr{F}_T)]$ est alors minimisée par cette intensité λ^*. Cette seconde *goodness-of-fit* n'est pas autant utilisée mais est plus simple à optimiser dans bien des cas. Ces deux mesures privilégient les fonctions d'intensité avec de hautes valeurs aux temps où les événements se produisent et des valeurs aussi faibles que possible à tous les autres temps.

1.2 Processus de Hawkes

Les processus de Hawkes [Haw71a, HO74] sont des processus ponctuels temporels dont l'intensité dépend de l'historique du processus avec un mécanisme d'excitation. Ils peuvent être considérés comme l'équivalent des modèles de séries temporelles autorégressives (AR) [Ham94] dans le cas où le temps est continu et non discret. Ils permettent d'analyser l'inter-causalité existante entre plusieurs séries d'événements. Ils ont d'abord été utilisés en séismologie [Oga99], puis en finance [BDHM13, BMM15] et ont aujourd'hui trouvé de nombreuses nouvelles applications comprenant la prédiction du crime [LMBB12, Moh13] ou la propagation de l'information dans les réseaux sociaux [CS08, BBH12, ZZS13a, YZ13, LSV+16].

Les processus de Hawkes multivariés modélisent les interactions de $D \geq 1$ processus ponctuels avec une dynamique d'excitation déterminée par la structure autorégressive de l'intensité conditionnelle. Pour chaque processus ponctuel $i = 1, \ldots, D$ cette intensité s'écrit

$$\lambda^i(t|\,\mathscr{F}_t) = \mu^i + \sum_{j=1}^{D} \int_0^t \varphi^{ij}(t-s)\,\mathrm{d}N_s^j.$$

Les $\mu^i \geq 0$ sont les intensités exogènes des nœuds $i = 1, \ldots, D$. Les φ^{ij} pour $1 \leq i, j \leq D$ sont appelés *noyaux*, ils quantifient, en magnitude et au cours du temps, l'influence

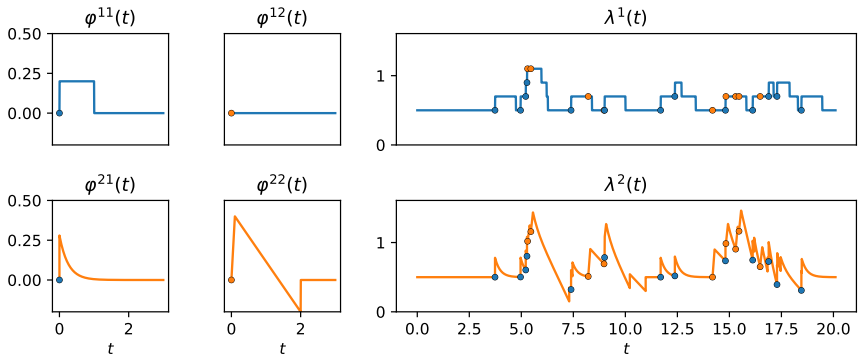

Figure I.1 : Une réalisation d'un processus de Hawkes de dimension deux. Les quatre noyaux sont montrés sur la partie de gauche. Les intensités sont affichés sur la partie de droite (en fonction du temps, jusqu'à $t = 20$), où les événements sont représentés par des points colorés (en bleu ceux qui correspondent au nœud 1, en orange au nœud 2).

des événements passés du nœud j sur l'intensité des événements du nœud i. La matrices des intégrales $(\Phi)_{1\leq i,j\leq D} = \int_0^T \varphi^{ij}(t)\,\mathrm{d}t$ renseigne l'espérance du nombre d'événements de type i directement provoqués par un événement de type j. Un processus de Hawkes admet un régime stationnaire quand le rayon spectral de cette matrice est inférieur à un : $\varrho(\Phi) < 1$ [BMM15]. De tels processus sont aisément simulés par l'algorithme de *thinning* [Oga81]. La Figure I.1 montre une réalisation d'un processus de Hawkes de dimension deux et met en évidence le mécanisme d'excitation en exhibant les fonctions de noyau φ^{ij} et l'impact qu'a eu chaque événement sur les intensités de chacun des nœud.

Estimation. Inférer un processus de Hawkes consiste à estimer son intensité exogène μ et ses fonctions de noyaux φ^{ij} de manière paramétrique ou non. Dans le cas non-paramétrique, les noyaux sont approchés par des histogrammes dans la plupart des cas [LM11, ZZS13b, BM14]. Cette procédure estime les noyaux du processus de manière très flexible mais passe difficilement à l'échelle. Récemment, [ABG+17] a proposé un nouvel estimateur non-paramétrique qui déduit directement la matrice Φ des intégrales des noyaux pour pallier ce problème. Dans le cas paramétrique, les estimateurs reposent sur une connaissance a priori des fonctions de noyaux qui sont alors décrites par un ensemble de paramètres. Estimer ces fonctions de noyaux revient donc à estimer ces paramètres. Ces procédures sont moins flexibles que les méthodes non-paramétriques mais sont plus efficaces et plus robustes car le nombre de paramètres à estimer est plus faible. Différentes paramétrisations sont utilisées pour modéliser des comportements variés, tels qu'une réponse retardée à un événement déclencheur

[XFZ16] ou une influence qui décroit lentement au cours du temps avec les noyaux de loi de puissance utilisés en séismologie [Oga88] et en finance. Cependant, pour des raisons de passage à l'échelle, la plupart des estimateurs paramétriques reposent sur des noyaux à décroissance exponentielle [ELL11, ZZS13a, TFSZ15, FWR+15, LSK17].

Noyaux exponentiels. La paramétrisation *exponentielle* des noyaux est le principal modèle paramétrique pour lequel nous considérons que $\varphi^{ij}(t) = \alpha^{ij}\beta\exp(-\beta t)$ pour $\alpha^{ij} > 0$ et $\beta > 0$ (voir le noyau φ^{21} dans la Figure I.1). Dans le modèle, la matrices des intégrales $\Phi = [\alpha^{i,j}]_{1\leq i,j\leq d}$ et $\beta > 0$ est un paramètre caractérisant la mémoire du processus. Le couple $(N_t, \lambda(t))$ est un processus de Markov [BMM15, Proposition 2], et l'équation de l'intensité conditionnelle se réécrit dans une forme Markovienne

$$\mathrm{d}\lambda^i(t|\ \mathscr{F}_t) = \sum_{j=1}^{D} \beta(\mu^j - \lambda^j(t|\ \mathscr{F}_t))\,\mathrm{d}t + \alpha^{ij}\beta\,\mathrm{d}N_s^j$$

pour $i = 1,\dots D$. Ainsi, au lieu de devoir considérer tous les événements passés, la valeur de l'intensité conditionnelle λ^i au temps t_2 peut être calculée à partir de sa valeur à un temps précédent $t_1 < t_2$ et les temps des événements qui ont eu lieu entre t_1 et t_2. Cette propriété permet des calculs beaucoup plus efficaces et un meilleur passage à l'échelle aussi bien pour la simulation que pour l'estimation.

Une approche plus générale est le noyau de *somme d'exponentielles* [LV14], soit $\varphi^{ij}(t) = \sum_{u=1}^{U} \alpha_u^{ij}\beta_u\exp(-\beta_u t)$ pour $\alpha_u^{ij} > 0$ et $\beta_u > 0$. Ces noyaux bénéficient encore de la propriété de Markov et se généralisent mieux car ils traitent avec plusieurs échelles de temps β_u à la fois et ainsi peuvent par exemple approcher des noyaux de loi-puissance [HBB13, FS15]. Pour des raisons de convexité, les paramètres de mémoire β sont généralement fixés pendant l'estimation [LV14]. Ensuite, afin de retrouver les paramètres μ et α, l'estimateur du maximum de vraisemblance, comme celui des moindres carrés, bénéficie de la propriété de Markov pour accélérer les calculs. En effet, leur complexité est linéaire en le nombre total d'événements N_T au lieu d'être quadratique comme dans le cas général. Bien que moins utilisé, l'estimateur des moindres carrés est une fonction quadratique qui est très simple à optimiser à l'inverse de l'estimateur du maximum de vraisemblance. Aussi, il passe très bien à l'échelle avec le nombre total d'événement car un niveau de précision aussi fin que souhaité peut être atteint avec un nombre fixe, $D^2 \times U^2$, de passages sur les données, voir le Chapitre I pour plus de détails.

Nous allons désormais nous focaliser sur la propagation de l'information dans les réseaux sociaux. C'est un problème stimulant à l'intérêt croissant [dMB04, Les08, CS08, LBK09] grâce au nombreuses applications dans la publicité ou le commerce en ligne où des enregistrements d'historiques d'événements de grande taille sont disponibles. Une approche supervisé habituelle consiste à prédire des étiquettes à partir

d'interactions déclarées (amitié, mention j'aime, compte suivi, etc.). Cependant, une telle supervision n'est pas toujours disponible, et elle ne décrit pas forcément précisément le niveau des interactions entre les utilisateurs. Les étiquettes sont souvent seulement binaires alors que la quantification d'une interaction est plus complexe et que les interactions déclarées souvent sont obsolètes. Plus généralement, une approche supervisée n'est pas suffisante pour retrouver les communautés latentes d'utilisateurs alors que les motifs temporels des actions des utilisateurs sont bien plus instructives. Avec les processus de Hawkes, nous considérons une approche directement conçue à partir des données inaltérées des actions des utilisateurs. Formellement, les utilisateurs sont les nœuds du processus ponctuel multivarié et les actions chronométrées sont les événements. Cependant, un processus de Hawkes brut retrouverait difficilement les motifs caractéristiques observés sur un réseau social tel que les interactions sparses entre les utilisateurs et les structures de communautés. Ceci soulève la question suivante,

Question 4. *Comment retrouver des dynamiques d'interaction sociales avec des processus de Hawkes ?*

Nous discutons de cette problématique dans la section suivante.

2 Processus de Hawkes multivariés, sparses et de faible rang

Nous considérons le problème consistant à dévoiler la structure implicite du réseau des interactions entre utilisateurs dans un réseau social, simplement à partir de séries d'événements en haute fréquence. Récemment, une approche conçue à partir des processus de Hawkes a gagné en popularité [CS08, BBH12, ZZS13a, YZ13]. Elle exploite la structure des processus de Hawkes pour retrouver l'influence directe qu'a une action d'un utilisateur spécifique sur toutes les futures actions des tous les utilisateurs (dont lui-même). Les processus de Hawkes modélisent simultanément le déclin de l'influence au court du temps avec la forme des noyaux φ^{ij}, les niveaux d'interactions entre les nœuds avec la matrice d'adjacence asymétrique et pondérée Φ et l'intensité exogène, qui mesure la spontanéité avec laquelle une action est réalisée sans l'influence des événements précédents.

2.1 Pénalisations ℓ_1 et norme trace

La pénalisation (aussi appelée régularisation) est une technique habituelle en apprentissage statistique. Elle consiste à minimiser l'opposé de la fonction de *goodness-of-fit*

plus un terme de pénalité façonné pour inculquer une structure spécifique à la solution du problème. La pénalisation Lasso ou ℓ_1 [Tib96] est une des plus connues et des plus utilisées. Dans un problème où $f : \mathbb{R}^d \to \mathbb{R}$ est l'opposé de la fonction de *goodness-of-fit*, la version pénalisée par ℓ_1 consiste à résoudre

$$\min_{w \in \mathbb{R}^d} f(w) + \lambda \|w\|_1,$$

où $\lambda > 0$ et $\|w\|_1 = \sum_{j=1}^d |w_j|$. Cette pénalité est connue pour donner des solutions avec un support sparse, c'est-à-dire un vecteur de coefficient w avec beaucoup d'entrées nulles. En fait, si nous considérons le cas plus général des pénalisation liées à la pénalisation ℓ_p, $\sum_{j=1}^d |w_j|^p$, alors le Lasso est la pénalisation ℓ_1 et la pénalisation ℓ_0, qui donne le nombre d'entrées non nulles dans un vecteur, est le cas limite quand $p \to 0$. Dans ce cadre, le Lasso emploie la plus petite valeur de p qui amène à une formulation convexe d'un problème pénalisé par ℓ_p. Ainsi, le lasso peut être vu comme la relaxation convexe du problème de sélection du meilleur sous-ensemble [Tib11].

La pénalisation *norme trace* (aussi appelée norme nucléaire) est utilisée quand les coefficients $\Omega \in \mathbb{R}^{d \times d}$ sont des matrices plutôt que des vecteurs. Elle consiste à ajouter à l'opposé de la fonction de *goodness-of-fit*, la norme trace de la matrice $\|\Omega\|_*$, donnée par

$$\|\Omega\|_* = \sum_{j=1}^d \sigma_j(\Omega),$$

où $\sigma_1(\Omega) \geq \cdots \geq \sigma_d(\Omega) \geq 0$ sont les valeurs singulières de Ω. Ainsi, elle est l'équivalent d'une pénalisation ℓ_1 appliquée au vecteur $[\sigma_1(\Omega) \ \cdots \ \sigma_d(\Omega)]$ et tend à favoriser l'apparition de valeurs singulières nulles. Finalement, comme le rang d'une matrice est égal au nombre de ses valeurs singulières non nulles, la pénalisation norme trace est une relaxation convexe du problème de faible rang [CT10], de la même manière que la pénalisation ℓ_1 l'est pour le problème de sélection du meilleur sous-ensemble. Favoriser un faible rang est habituel en filtrage collaboratif [CT04, CT10, RRS11] pour décrire la structure du réseau avec un nombre limité de paramètres. Rendu populaire par le prix Netflix [BL07], cela tend à faire apparaître des communautés d'individus au comportement similaire.

2.2 A priori sparse et de faible rang

Nous combinons ces pénalisations avec un processus de Hawkes à d nœuds, chacun d'entre eux représentant un utilisateur. Pour plus de simplicité, nous considérons dans ce chapitre que, pour tout $j, j' = 1, \dots, d$, les noyaux $\varphi^{jj'}$ peuvent être décomposés en $\varphi_{j,j'}(t) = \sum_{k=1}^K a_{j,j',k} h_{j,j',k}(t)$ où $a_{j,j',k} \geq 0$ and $h_{j,j',k} : \mathbb{R}^+ \to \mathbb{R}^+$ sont des fonctions de déclin connues et avec une norme ℓ_1 fixée, $\|h_{j,j',k}\|_1 = 1$.

Ainsi, pour chaque nœud $j = 1, \dots, d$, l'intensité conditionnelle s'écrit

$$\lambda_{j,\mu,\mathbb{A}}(t) = \mu_j + \sum_{j'=1}^{d} \sum_{k=1}^{K} a_{j,j',k} h_{j,j',k}(t-s)\, \mathrm{d}N_s^j,$$

où $\mu \in \mathbb{R}^d$ est l'intensité exogène. Par exemple, choisir $h_{j,j',k} = \beta_k \exp(-\beta_k t)$ où $\beta_k > 0$ revient à la paramétrisation en somme d'exponentielle (voir Section 1.2). Le paramètre d'intérêt est le tenseur *d'auto-excitation* $\mathbb{A}$ qui s'écrit simplement $\mathbb{A} = [a_{j,j',k}]_{1 \le j,j' \le d, 1 \le k \le K}$. Ensuite, nous combinons les a priori sparse et de faible rang du tenseur d'auto-excitation et de l'intensité exogène pour obtenir la structure du réseau désirée. Nos hypothèses sur μ et $\mathbb{A}$ sont les suivantes.

μ est sparse. Certains nœuds sont globalement inactifs et ne réagissent seulement s'ils sont stimulés. Ainsi, nous supposons que l'intensité exogène μ est sparse.

$\mathbb{A}$ est sparse. Un nœud n'interagit qu'avec une fraction des autres nœuds, ce qui signifie que pour un nœud fixé j, seuls quelques sont $a_{j,j',k}$ non nuls. De plus, un nœud peut ne réagir qu'à différentes échelles de temps, précisément, $a_{j,j',k}$ est non nul pour quelques k seulement dans un couple j, j' fixé. C'est pourquoi, nous supposons que les entrées du tenseur $\mathbb{A}$ sont sparses.

$\mathbb{A}$ est de faible rang. Nous supposons qu'il existe des facteurs latents qui expliquent comment les nœuds s'impactent les uns les autres à travers les différentes échelles de temps $k = 1, \dots, K$. En réécrivant l'intensité, cela mène naturellement à pénaliser le rang de la matrice $d \times Kd$ $\mathrm{hstack}(\mathbb{A}) = \left[\mathbb{A}_{\bullet,\bullet,1} \cdots \mathbb{A}_{\bullet,\bullet,K}\right]$ où $\mathbb{A}_{\bullet,\bullet,k}$ dénote la matrice $d \times d$ qui a pour entrées $(\mathbb{A}_{\bullet,\bullet,k})_{j,j'} = \mathbb{A}_{j,j',k}$.

Nous induisons ces a priori avec des termes de pénalisations ajoutés à l'objectif. Précisément, nous minimisons

$$R(\mu, \boldsymbol{A}) + \|\mu\|_{1,\hat{w}} + \|\mathbb{A}\|_{1,\hat{\mathbb{W}}} + \hat{\tau} \|\mathrm{hstack}(\mathbb{A})\|_*, \tag{A.2}$$

où $R(\mu, \boldsymbol{A})$ est la goodness-of-fit des moindres carrés (A.1), et les termes de pénalisation sont la norme trace et les pénalisations ℓ_1 pondérées, données par

$$\|\mu\|_{1,\hat{w}} = \sum_{j=1}^{d} \hat{w}_j |\mu_j|, \quad \|\mathbb{A}\|_{1,\hat{\mathbb{W}}} = \sum_{1 \le j,j' \le d, 1 \le k \le K} \hat{\mathbb{W}}_{j,j',k} |\mathbb{A}_{j,j',k}|.$$

Les poids $\hat{w}$, $\hat{\mathbb{W}}$, et le coefficient $\hat{\tau}$ sont des paramètres ajustés en fonction des données. Ils sont formalisés dans la Section 4 du Chapitre II. Le choix de ces poids permet, à partir des données, de tenir de la variation d'information disponible pour chaque nœud et provient d'une analyse fine des termes de bruit présenté dans la section ci-dessous.

2.3 Inégalité d'oracle

Avec un choix judicieux des paramètres de poids $\hat{w}$, $\hat{\mathbb{W}}$ et $\hat{\tau}$, nous obtenons, dans le Théorème 1 une inégalité d'oracle. Cette inégalité borne l'erreur d'estimation de l'intensité $\lambda_{\hat{\mu},\hat{\mathbb{A}}}$, obtenue en minimisant le Problème (A.2), étant donné le meilleur estimateur, avec la même paramétrisation, qui serait obtenu avec une information parfaite. Nous fixons un niveau de confiance $x > 0$, qui peut être choisi sans risque à $x = \log T$ par exemple, et dénommons par $\|\cdot\|_F$ la norme de Frobenius pour formuler le théorème suivant où aucune hypothèse n'est faite sur l'intensité réelle λ.

Théorème 1. *Fixons $x > 0$, et choisissons $\hat{w}, \hat{\boldsymbol{W}}, \hat{\tau}$ qui dépendent de x tels que données par* (II.17), (II.18) *et* (II.19). *Alors l'inégalité*

$$\|\lambda_{\hat{\mu},\hat{\mathbb{A}}} - \lambda\|_T^2 \leq \inf_{\mu,\mathbb{A}} \Big\{ \|\lambda_{\mu,\mathbb{A}} - \lambda\|_T^2 + 1.25\kappa(\mu,\mathbb{A})^2 \Big(\|(\hat{w})_{\mathrm{supp}(\mu)}\|_2^2 + \|(\hat{\mathbb{W}})_{\mathrm{supp}(\mathbb{A})}\|_F^2 + \hat{\tau}^2 \,\mathrm{rank}(\mathrm{hstack}(\mathbb{A})) \Big) \Big\}$$

est vérifiée avec une probabilité supérieure à $1 - 70.35e^{-x}$.

La constante $\kappa(\mu,\mathbb{A})$ est donnée par la Définition 1 du Chapitre II. Elle vient de la nécessité d'avoir une condition de restriction des valeurs propres de la matrice de Gram du problème pour obtenir une inégalité d'oracle avec un taux rapide [BRT09, Koll1]. Grossièrement, elle demande que pour tout ensemble de paramètres $\{\mu', \mathbb{A}'\}$ qui a un support proche de celui de $\{\mu, \mathbb{A}\}$, nous ayons que la norme L^2 de $\{\mu', \mathbb{A}'\}$ dans le support de $\{\mu, \mathbb{A}\}$ puisse être bornée par la norme L^2 de l'intensité donnée par $\|\lambda_{\mu',\mathbb{A}'}\|_T$.

2.4 Expériences numériques

Pour mesurer les performances de ces pénalisations pondérées à partir de des données $\{\hat{w}, \hat{\mathbb{W}}, \hat{\tau}\}$, nous menons une suite d'expérience sur des jeux de données synthétiques et comparons notre méthode aux pénalisations non pondérées [ZZS13a]. Nous réalisons ces expériences sur des processus de Hawkes avec $d = 30$ nœuds et $K = 3$ bases de noyaux où le tenseur d'auto-excitation contient des boites se chevauchant, correspondant aux communautés, afin de respecter les a priori sparse et de faible rang. Nous considérons quatre méthodes d'estimation qui minimisent l'erreur des moindres carrées à laquelle s'ajoute l'une des pénalisations suivantes :

- L1 : Pénalisation ℓ_1 non pondérée de $\mathbb{A}$
- wL1 : Pénalisation ℓ_1 pondérée de $\mathbb{A}$

- L1Nuclear : Pénalisation ℓ_1 non pondérée et norme trace de hstack($\mathbb{A}$) (identique à [ZZS13a])

- wL1Nuclear : Pénalisation ℓ_1 pondérée et norme trace de hstack($\mathbb{A}$)

Ensuite, pour chaque procédure, nous entraînons le modèle sur des données générées, en le limitant sur un intervalle de temps croissant, et évaluons ses performances pour chaque temps avec les trois mesures suivantes :

- Erreur d'estimation : l'erreur d'estimation relative ℓ_2 de $\mathbb{A}$, donnée par $\|\hat{\mathbb{A}} - \mathbb{A}\|_2^2 / \|\mathbb{A}\|_2^2$

- AUC : Nous calculons l'AUC (aire sous la courbe ROC) entre la matrice génératrice $\mathbb{A}$ binarisée et la solution $\hat{\mathbb{A}}$ dont les entrées sont réechelonnées dans $[0, 1]$. Ceci permet de quantifier la capacité de la procédure à retrouver le support de la structure de la connexion entre les nœuds.

- Kendall : Nous calculons le tau-b de Kendall entre toutes les entrées de la matrice génératrice $\mathbb{A}$ et la solution $\hat{\mathbb{A}}$. Ce coefficient de corrélations prend des valeurs entre -1 et 1 et compare le nombre de paires concordantes et discordantes. Ceci nous permet de quantifier la capacité de la procédure à classer correctement l'intensité de la connexion entre les nœuds.

La Figure I.2 confirme que les pénalisations pondérées amènent systématiquement une amélioration, pour L1 et L1Nuclear, en termes d'erreur d'estimation, d'AUC et de coefficient de Kendall.

Étudier les techniques d'optimisation utilisées pour minimiser l'objectif (A.2) allait au delà du périmètre de cette section. Cependant, l'optimisation est une partie cruciale de la procédure sur laquelle nous allons nous concentrer dans les deux sections suivantes.

3 Minimisation de sommes composites avec des méthodes du premier ordre

Une grande variété de tâches en apprentissage automatique consistent à optimiser le problème suivant

$$\min_{w \in \mathbb{R}^d} F(w) \quad \text{avec} \quad F(w) = f(w) + g(w), \quad f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w),$$

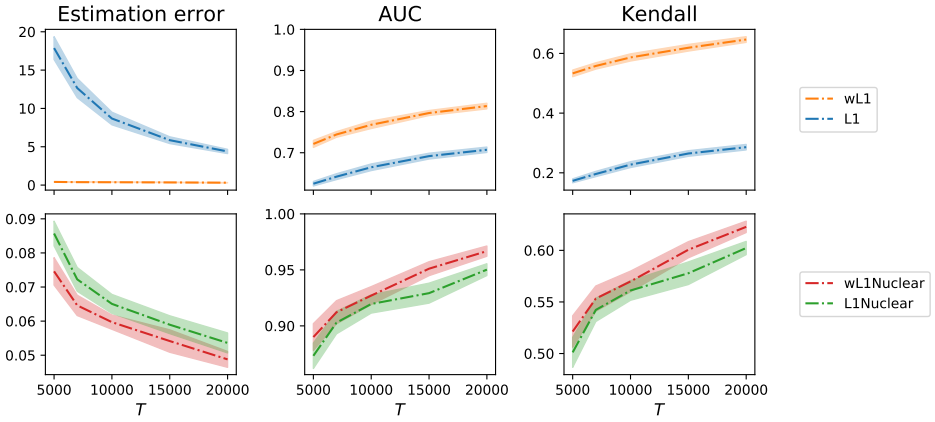

Figure I.2 : Valeur des mesures pour des données simulées de dimension $d = 30$ et avec $K = 3$ bases de noyaux. L'abscisse correspond à la longueur de l'intervalle R. Les pénalisations pondérées amènent systématiquement une amélioration, à la fois pour les pénalisations L1 et L1Nuclear.

où les fonctions f_i correspondent à une perte calculée à l'observation i du jeu de données et la fonction convexe g est un terme de pénalisation. Ce cadre inclut la classification au moyen de la régression logistique avec $f_i(w) = \log(1 + \exp(-y_i w^\top x_i))$, la régression des moindres carrés avec $f_i(w) = (y_i - w^\top x_i)^2$ parmi beaucoup d'autres. Il est habituel de supposer que la fonction f est gradient-Lipschitz, soit $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ pour tout $x, y \in \mathbb{R}^d$ où $\|.\|$ dénote la norme euclidienne sur $\mathbb{R}^d$, et $L > 0$ est la constante de Lipschitz. Avec cette propriété, le *lemme de descente* [Ber99, Proposition A.24] est vérifié,

$$f(w + \Delta w) \leq f(w) + \Delta w^\top \nabla f(w) + \frac{L}{2}\|\Delta w\|^2$$

pour tout $w, \Delta w \in \mathbb{R}^d$. La plupart des algorithmes du premier ordre découlent de ce lemme et en premier lieu, l'algorithme de descente de gradient où à chaque itération Δw^{t+1} est définit par la valeur optimale $-\frac{1}{L}\nabla f(w^t)$. Le terme de pénalisation est alors géré avec les opérateurs proximaux [CP11]. C'est ainsi que fonctionnent l'algorithme ISTA et sa version accélérée FISTA [BT09] dont la vitesse de convergence est optimale [Nes83].

Descente de gradient stochastique. Cependant, avec sa structure, ce problème peut également être considéré comme une accumulation de plus petits problèmes f_i qui ont un comportement commun. La descente de gradient stochastique (SGD) [RM51] l'exploite et, au lieu de calculer le gradient complet $\nabla f(w^t)$ à chaque itération, emploie une variable aléatoire $\phi_t \in \mathbb{R}^d$ telle que $\mathbb{E}[\phi_t] = \nabla f(w^t)$. L'étape de descente

devient $\Delta w^{t+1} = -\eta_t \phi_t$ où ϕ_t vaut $\nabla f_i(w^t)$ et $\eta_t > 0$ est un pas de descente. Quand tous les $\nabla f_i(w^t)$ sont aussi coûteux à calculer, SDG réalise des itérations qui sont n fois plus rapides que les algorithmes par paquets introduits précédemment. Mais ces méthodes ne convergent pas facilement vers une solution précise car $\nabla f_i(w^t)$ n'approche pas zéro quand w^t est proche de la valeur optimale w^*. Ainsi, la suite des pas de descente $(\eta_t)_{t\geq 0}$ doit être décroissante ce qui affecte, en définitive, la vitesse de convergence.

Algorithmes stochastiques avec réduction de variance. Récemment, des algorithmes stochastiques fondés sur une combinaison de SGD avec la technique de réduction de variance de Monte-Carlo [SLRB17, SSZ13, JZ13, DBLJ14] s'avèrent être à la fois très efficaces numériquement (chaque itération a une complexité comparable à celle de SGD) et très solides théoriquement. Pour réduire sa variance, la variable aléatoire ϕ_t prend pour valeur $\nabla f_i(w^t) + Y - \mathbb{E}[Y]$ où $Y \in \mathbb{R}^d$ est une autre variable aléatoire dont on attend qu'elle soit corrélée à $\nabla f_i(w^t)$. Ainsi, ϕ_t demeure un estimateur non biaisé de $\nabla f(w^t)$ et sa variance est diminuée. Dans [SLRB17, SSZ13, JZ13] et [DBLJ14], ϕ^t converge vers 0 à l'optimum, et la séquence des pas de descente $(\eta_t)_{t\geq 0}$ n'a plus besoin d'être décroissante comme pour SGD. Ces algorithmes obtiennent un taux de convergence linéaire, ce qui signifie qu'ils atteignent un itéré w^t tel que $F(w^t) \leq F(w^*) + \varepsilon$ en moins de $\mathcal{O}(\log(1/\varepsilon))$ itérations.

Ainsi, les techniques d'optimisation modernes obtiennent des solutions très précises avec peu de passages sur les données. Cependant, les algorithmes du premier ordre que nous venons d'introduire reposent sur l'hypothèse que f est gradient-Lipschitz ce qui n'est pas vérifié par la log-vraisemblance des processus de Hawkes. Le manque d'une méthode rapide, robuste et passant à l'échelle pour résoudre ce problème motive la question suivante.

Question 5. *Comment optimiser des objectifs non gradient-Lipschitz tels que log-vraisemblance des processus de Hawkes ?*

Nous nous focalisations sur ce problème particulier dans la section suivante où nous développons un algorithme dédié à une nouvelle classe de fonction, admettant une nouvelle propriété de régularité.

4 Optimisation duale pour les objectifs convexes contraints sans l'hypothèse de gradient-Lipschitz

Quand l'hypothèse de gradient-Lipschitz n'est pas vérifiée, le lemme de descente ne tient plus et les algorithmes précédents n'ont plus de garanties de convergence.

Motivés par l'étude de problèmes qui ne vérifient pas cette hypothèse, tels que la régression de Poisson linéaire et les processus de Hawkes, nous travaillons avec une autre hypothèse de régularité, et obtenons un taux de convergence linéaire pour une version décalée de SDCA [SSZ13] qui améliore l'état de l'art actuel.

SDCA pour les objectifs log **réguliers**. Afin de s'affranchir de l'hypothèse de gradient-Lipschitz, nous devons nous atteler à une tâche plus spécifique dépendant d'une nouvelle hypothèse de régularité. Pour des fonctions convexe $f_i : \mathscr{D}_f \to \mathbb{R}$ où $\mathscr{D}_f = (0, +\infty)$ telles que $\lim_{t\to 0} f_i(t) = +\infty$, $\psi \in \mathbb{R}^d$, $x_1, \dots, x_n \in \mathbb{R}^d$, $\lambda > 0$ et avec une fonction $g : \mathbb{R}^d \to \mathbb{R}$ 1-fortement convexe, nous considérons l'objectif

$$\min_{w \in \Pi(X)} P(w) \quad \text{où} \quad P(w) = \psi^\top w + \frac{1}{n}\sum_{i=1}^n f_i(w^\top x_i) + \lambda g(w), \tag{A.3}$$

où $\Pi(X)$ est le polytope $\{w \in \mathbb{R}^d \ : \ \forall i \in \{1, \dots, n\},\ w^\top x_i > 0\}$ que nous supposons être non vide. Les algorithmes du premier ordre précédemment introduits n'ont pas de garanties théoriques pour ce problème et ne parviennent pas à maintenir leur itérés w^t dans le polytope $\Pi(X)$ au cours de nos expériences. Pour travailler avec des contraintes plus simples, nous nous concentrons plutôt sur le problème dual qui a simplement des contraintes boites,

$$\max_{\alpha \in -\mathscr{D}_{f^*}^n} D(\alpha) \quad \text{où} \quad D(\alpha) = \frac{1}{n}\sum_{i=1}^n -f_i^*(-\alpha_i) - \lambda g^*\left(\frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i - \frac{1}{\lambda}\psi\right),$$

où f^* (resp. g^*) est la conjuguée de Fenchel de f (resp. g) et $-\mathscr{D}_{f^*}^n$ est le domaine de la fonction $x \mapsto \sum_{i=1}^n f_i^*(-x)$. Alors que la forte dualité n'est pas directement garantie pour un problème convexe avec des contraintes ouvertes, elle est bien vérifiée dans ce contexte (voir la Proposition 1 du Chapitre IV). Ainsi, nous maximisons ce dual avec une variante décalée de l'algorithme SDCA [SSZ13] qui ne repose pas sur l'hypothèse de gradient-Lipschitz mais plutôt sur la propriété suivante de log régularité.

Définition 1. *Une fonction $f : \mathscr{D}_f \subset \mathbb{R} \to \mathbb{R}$ est dite* L-log régulière, *où $L > 0$, s'il s'agit d'une fonction convexe, différentiable et strictement monotone telle que*

$$|f'(x) - f'(y)| \leq \frac{1}{L} f'(x) f'(y) |x - y|$$

pour $\forall x, y \in \mathscr{D}_f$.

En réalité, à la lumière de la proposition suivante, la log régularité est liée à la propriété de self-concordance, introduite par Nesterov [Nes13] et largement utilisée pour étudier des fonctions impliquant des logarithmes.

Proposition 1. *Soit $f : \mathcal{D}_f \subset \mathbb{R} \to \mathbb{R}$ une fonction convexe, strictement monotone et deux fois différentiable. Alors*

$$f \text{ est } L\text{-log } \textit{régulière} \quad \Leftrightarrow \quad \forall x \in \mathcal{D}_f,\ f''(x) \leq \tfrac{1}{L} f'(x)^2.$$

Il apparaît que la log régularité est le pendant de la self-concordante mais pour contrôler le second ordre avec la dérivée du premier ordre. En supposant que toutes les fonctions f_i sont L_i-log régulières, nous dérivons de nouvelles inégalités de convexité et prouvons dans le Chapitre IV le théorème suivant où $\alpha^* \in \mathbb{R}^n$ est l'optimum de l'objectif dual.

Théorème 2. *Connaissant des bornes $\beta_i \in -\mathcal{D}_{f^*}^n$ telles que $R_i = \frac{\beta_i}{\alpha_i^*} \geq 1$ pour $i = 1, \ldots, n$ et en supposant que tous les f_i sont L_i-log régulières et g est 1-fortement convexe. Alors SDCA satisfait*

$$\mathbb{E}[D(\alpha^{(t)}) - D(\alpha^*)] \geq \left(1 - \frac{\min_i \sigma_i}{n}\right)^t \left(D(\alpha^*) - D(\alpha^{(0)}\right),$$

où

$$\sigma_i = \left(1 + \frac{\|x_i\|^2 \alpha_i^{*2}}{2\lambda n L_i} \frac{(R_i - 1)^2}{\frac{1}{R_i} + \log R_i - 1}\right)^{-1}.$$

Ce théorème donne un taux de convergence linéaire pour l'objectif dual qui améliore ce qui est obtenu avec l'analyse standard de SDCA. Ensuite, nous améliorons ces garanties théoriques en proposant une variante avec de l'échantillonnage préférentiel et son efficacité numérique avec une heuristique pour l'initialisation et une méthode de mini-lot.

Application à la régression de Poisson et aux processus de Hawkes La régression de Poisson dite *linéaire* est largement utilisée pour la reconstruction d'images [HMW12], le marketing sur internet [CPC09] et en analyse de survie pour modéliser des effets additifs plutôt que multiplicatifs [BF10]. SDCA pour les objectifs log réguliers s'applique à la régression de Poisson linéaire ainsi qu'aux processus de Hawkes avec sommes d'exponentielles précédemment introduits. Pour les deux modèles, nous pouvons formuler leur vraisemblance comme dans l'Équation (A.3) et donner des candidats explicites pour les bornes β_i nécessaires pour le Théorème 2. Alors que la formulation du problème est directe pour la régression de Poisson, elle implique des poids précalculés pour les processus de Hawkes et amène à I sous problèmes indépendants où I désigne le nombre de nœuds du processus de Hawkes.

Expérimentalement, nous comparons SDCA avec un algorithme du second ordre, la version standard de l'algorithme de Newton qui calcule à chaque itération la hessienne de l'objectif qui est alors utilisée pour résoudre un système linéaire. Cela

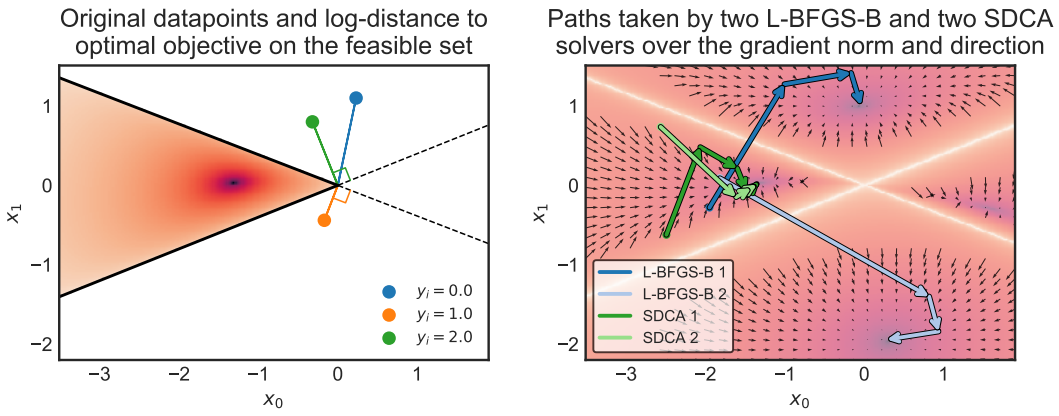

Figure I.3 : Itérés de SDCA et L-BFGS-B pour une régression de Poisson sur un exemple jouet avec trois observations de deux variables. *Gauche.* Jeu de données et valeur de l'objectif. *Droite.* Itérés de L-BFGS-B et SDCA avec deux points de départ différents. L'arrière plan représente la norme du gradient et les flèches sa direction. SDCA est très stable et converge rapidement vers l'optimum alors que L-BFGS-B converge facilement en dehors du domaine de définition de l'objectif.

garantit une convergence supra-linaire et maintient tous les itérés dans le polytope ouvert $\Pi(X)$, à partir du moment où le point de départ y est également [NN94]. Cependant, cet algorithme passe très mal à l'échelle en particulier quand la dimension d (la longueur du vecteur w) augmente. Cela limite drastiquement son utilisation en pratique. Ensuite, nous comparons SDCA avec SVRG [JZ13, TMDQ16] et l'algorithme quasi-Newton à mémoire limitée L-BFGS-B [Noc80, NW06]. Tous deux reposent théoriquement sur l'hypothèse du gradient-Lipschitz qui n'est pas vérifiée dans notre cas. En pratique, ils divergent très souvent et violent la contrainte du polytope $\Pi(X)$. Ceci est bien illustré dans l'exemple jouet de la Figure I.3. Ainsi, afin d'obtenir des résultats comparables, nous avons dû forcer la contrainte en projetant les itérés de SVRG et L-BFGS-B dans $[0, +\infty)^d$.

Comme attendu, dans la Figure I.4, nous observons que l'algorithme de Newton devient très lent lorsque le nombre de variables d augmente et, que SVRG et L-BFGS-B ne peuvent pas atteindre la solution optimale car leurs itérés sont contraints dans $[0, +\infty)^d$ alors que la solution contient des valeurs négatives. SDCA est le seul algorithme du premier ordre qui atteint la solution et combine le meilleur des deux mondes : la vitesse et la scalabilité des algorithmes du premier ordre avec la capacité à trouver des solutions comprenant des valeurs négatives.

Les processus de Hawkes et l'optimisation convexe d'objectifs de sommes finies

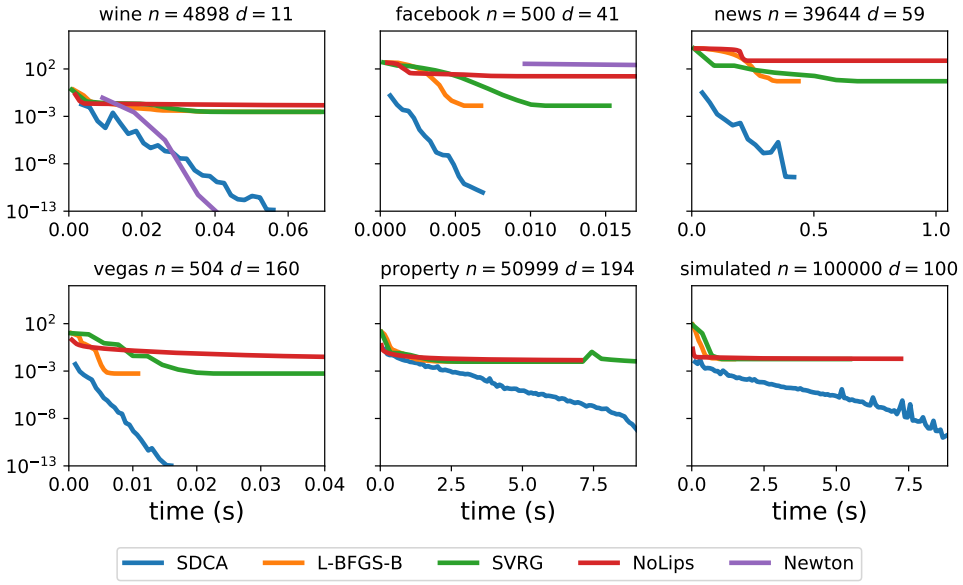

Figure I.4 : Convergence au cours du temps de quatre algorithme SDCA, SVRG, L-BFGS-B et Newton sur six jeux de données de régression de Poisson. SDCA combine le meilleur des deux mondes : la vitesse et la scalabilité de SVRG et L-BFGS-B avec la même précision que l'algorithme de Newton.

sont deux champs d'intérêt croissant. Dans les deux cas, les résultats numériques sont de première importance mais les articles sont rarement publiés avec un code qui permet non seulement de reproduire les résultats mais qui est également pensé pour être réutilisé pour des applications futures. Cela rend les algorithmes d'optimisation convexe et les estimateurs de processus de Hawkes difficiles à comparer de manière unifiée et soulève la question suivante.

Question 6. *Comment rendre disponible au plus grand nombre ces outils d'inférence statistique ?*

Dans la section suivante, nous présentons une nouvelle bibliothèque Python traitant à la fois l'optimisation convexe et les processus de Hawkes afin de faciliter leur utilisation pratique.

Tableau I.1 : Modèles et techniques d'estimation pour les processus de Hawkes disponibles dans `tick`

Non Paramétrique	Paramétrique
EM [LM11]	Noyaux exponentiels simples
Noyaux à bases [ZZS13a]	Somme de noyaux exponentiels
Wiener-Hopf [BM14]	Somme de noyaux gaussiens [XFZ16]
NPHC [ABG+17]	ADM4 [ZZS13a]

5 tick : une bibliothèque Python pour l'apprentissage statistique

`tick` est une bibliothèque d'apprentissage statistique pour Python 3 qui traite en particulier les modèles dépendant du temps, tels que les processus ponctuels, des outils pour les modèles linéaires généralisés et l'analyse de survie. Elle s'appuie sur une implémentation en C++ et les algorithmes d'optimisation issus de l'état de l'art pour réaliser des calculs rapides dans un environnement multicœur. Le code source et la documentation peuvent être téléchargés à partir de https://github.com/X-DataInitiative/tick.

Hawkes En dépit de l'intérêt croissant pour les processus de Hawkes, très peu de bibliothèques sont disponibles pour les étudier. Il en existe trois principales. La bibliothèque `pyhawkes`[1] propose un petit échantillon de modèles d'inférence bayésienne pour les processus de Hawkes. `hawkes` R[2] est une bibliothèque basée sur R et qui ne dispose que d'un seul algorithme d'estimation peu optimisé. Enfin, `PtPack`[3] est une bibliothèque C++ qui comprend des estimateurs du maximum de vraisemblance pour des modèles paramétriques avec des pénalisations ℓ_1. Écrite en Python, `tick` est la bibliothèque la plus complète traitant des processus de Hawkes. Elle inclut, par exemple, les principaux algorithmes d'inférence de la littérature listés dans le Tableau I.1 qui comprend les algorithmes aussi bien paramétriques que non paramétriques et les rend plus accessibles qu'auparavant.

Boite à outils pour l'optimisation convexe Au-delà des processus de Hawkes, `tick` contient trois modules principaux : `tick.linear_model` avec les régressions linéaire, logistique et de Poisson, `tick.robust` pour des modèles linéaires robustes

[1] https://github.com/slinderman/pyhawkes
[2] https://cran.r-project.org/web/packages/hawkes/hawkes.pdf
[3] https://github.com/dunan/MultiVariatePointProcess

Tableau I.2 : `tick` permet à l'utilisateur de combiner de nombreux modèles, prox et résolveurs. Cette liste n'est pas exhaustive.

Modèle	Op. proximal	Résolveur
Régression linéaire	SLOPE	Descente de gradient
Régression logistique	L1 (Lasso)	Descente de gradient accélérée
Régression de Poisson	Total Variation	Descente de gradient stochastique
Régression de Cox	Group L1	SVRG [JZ13]
Hawkes with exp. kernels	L2 (Ridge)	SDCA [SSZ13]

et `tick.survival` pour l'analyse de survie. À haut niveau, `tick` suit l'API de `scikit-learn` [PVG+11, BLB+13] qui est bien connue pour son caractère complet et sa facilité d'utilisation. Mais, derrière, ces modules reposent sur une boite à outil d'optimisation pour résoudre le problème de minimisation des sommes composites (3). Cette boite à outils permet de combiner, avec beaucoup de modèles, différentes techniques de pénalisation (module `tick.prox`) et les derniers algorithmes d'optimisation convexe (`tick.solver`). Tout ceci est implémenté de manière très modulaire et permet plus de possibilités que les autres bibliothèques d'optimisation basés sur l'API de `scikit-learn` telles que `lightning`[4]. Une liste non exhaustive des combinaisons possibles est donnée dans le Tableau I.2. Elle souligne à quel point `tick` est utile pour faire des expériences consistant par exemple à tester un nouveau modèle avec diverses techniques de pénalisation ou à comparer plusieurs algorithmes d'optimisation convexe.

Implémentation Bien que `tick` soit une bibliothèque Python, tous les calculs lourds sont exécutés en C++ qui communique alors ses résultats à Python avec SWIG (Simplified Wrapper and Interface Generator) [BFK+96]. Grâce à SWIG, les objets Python ont facilement accès à des objets C++ purs avec lesquels ils partagent leur mémoire et qui travaillent donc sur les mêmes jeux de données sans nécessiter aucune copie. Ceci est particulièrement utile pour l'optimisation où les objets `model`, `prox` et `solver` sont symboliquement associés en Python et ensuite exécutent tous leurs calculs en C++. Aussi, la partie C++ de la bibliothèque est indépendante et est utilisable sans Python au prix de quelques efforts. Cela permet aux développeurs d'analyser le code avec n'importe quel outil de *profiling* compatible avec C++ et ainsi de produire du code optimisé en profondeur. Pour toutes ces raisons, `tick` est une bibliothèque très efficace et a prouvé qu'elle était plus rapide (jusqu'à un ordre de grandeur) que `hawkes` R et `PtPack` sur une série d'expériences standard présentées dans la Figure I.5.

[4] http://contrib.scikit-learn.org/lightning

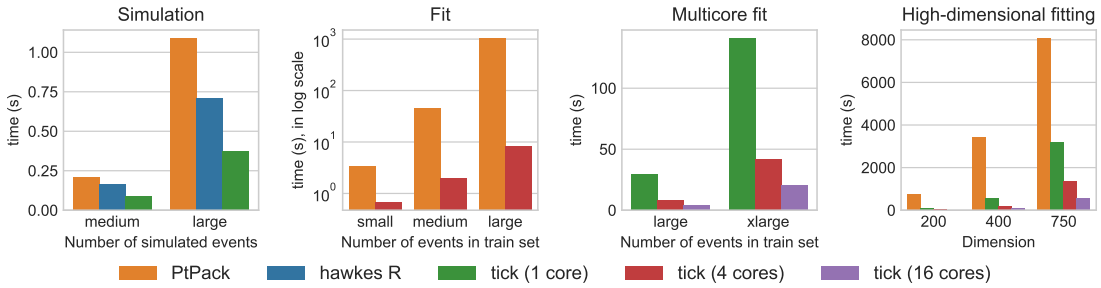

Figure I.5 : Temps de calcul de `tick` comparé à `PtPack` et `hawkes` R. `tick` surpasse sensiblement les deux bibliothèques pour la simulation et l'estimation. (Noter que le graphique "Fit" est en échelle logarithmique). Les temps d'apprentissage des graphiques "Fit" et "Multicore fit" sont comparés sur une simulation d'un processus de Hawkes de 16 dimensions avec un nombre croissant d'événements, small=5×10^4, medium=2×10^5, large=10^6, xlarge= 5×10^7, alors que des processus de Hawkes de dimension 200, 400 et 750 sont estimés dans le graphique "High-dimensional fitting". Les graphiques "Multicore fit" et "High-dimensional fitting" montrent que `tick` bénéficie des environnements à plusieurs cœurs pour accélérer ses calculs.

université PARIS-SACLAY | École doctorale de mathématiques Hadamard (EDMH)

Titre : Apprentissage automatique avec les processus de Hawkes et l'optimisation stochastique

Mots clés : Processus de Hawkes, optimisation stochastique, causalité, logiciel libre

Résumé : Le fil rouge de cette thèse est l'étude des processus de Hawkes. Ces processus ponctuels décryptent l'inter-causalité qui peut avoir lieu entre plusieurs séries d'événements. Concrètement, ils déterminent l'influence qu'ont les événements d'une série sur les événements futurs de toutes les autres séries. Par exemple, dans le contexte des réseaux sociaux, ils décrivent à quel point l'action d'un utilisateur, tel un Tweet, sera susceptible de déclencher des réactions de la part des autres.
Le premier chapitre est une brève introduction sur les processus ponctuels suivie par un approfondissement sur les processus de Hawkes et en particulier sur les propriétés de la paramétrisation à noyaux exponentiels, la plus communément utilisée. Dans le chapitre suivant, nous introduisons une pénalisation adaptative pour modéliser, avec des processus de Hawkes, la propagation de l'information dans les réseaux sociaux dont nous connaissons les caractéristiques a priori. Notre technique utilise des pénalités pondérées dont les poids sont déterminés par une analyse fine de l'erreur de généralisation.
Ensuite, nous traitons de l'optimisation convexe et des progrès réalisés avec les méthodes stochastiques du premier ordre avec réduction de variance. Le quatrième chapitre est dédié à l'adaptation de ces techniques aux processus de Hawkes. En effet, leur fonction de perte ne vérifie pas l'hypothèse de gradient-Lipschitz habituellement utilisée. Ainsi, nous travaillons avec une autre hypothèse de régularité, et obtenons un taux de convergence linéaire. De plus, notre algorithme respecte les contraintes linéaires du modèle ce qui pourtant requiert habituellement des méthodes du second ordre dont le coût est prohibitif en grandes dimensions.
Enfin, le dernier chapitre présente une nouvelle bibliothèque d'apprentissage statistique pour Python 3 avec un accent particulier mis sur les modèles temporels. Appelée tick, cette bibliothèque repose sur une implémentation en C++ et des algorithmes d'optimisation issus de l'état de l'art pour réaliser des estimations très rapides dans un environnement multi-cœurs. Publiée sur Github, cette bibliothèque a été utilisée pour réaliser les expériences de cette thèse.

Title : Machine learning based on Hawkes processes and stochastic optimization

Keywords : Hawkes processes, stochastic optimization, causality, open source

Abstract : The common thread of this thesis is the study of Hawkes processes. These point processes decrypt the cross-causality that occurs across several event series. Namely, they retrieve the influence that the events of one series have on the future events of all series. For example, in the context of social networks, they describe how likely an action of a certain user (such as a Tweet) will trigger reactions from the others.
The first chapter consists in a general introduction on point processes followed by a focus on Hawkes processes and more specifically on the properties of the widely used exponential kernels parametrization. In the following chapter, we introduce an adaptive penalization technique to model the information propagation on social networks with Hawkes processes. This penalization is able to take into account the prior knowledge on the social network characteristics. Our technique uses data-driven weighted penalties induced by a careful analysis of the generalization error.
Next, we focus on convex optimization and recall the recent progresses made with stochastic first order methods using variance reduction techniques. The fourth chapter is dedicated to an adaptation of these techniques to Hawkes processes. Indeed, their loss function does not meet the gradient-Lipschitz assumption that is required by the latest first order methods. Thus, we work under another smoothness assumption and obtain a linear convergence rate. Besides, our algorithm is compatible with the linear constraints of the model even though it usually requires second order methods that are very expensive in high dimensions.
Finally, the last chapter introduces a new statistical learning library for Python 3 with a particular emphasis on time-dependent models. Called tick, this library relies on a C++ implementation and state-of-the-art optimization algorithms to provide very fast computations in a single node multi-core setting. Open-sourced and published on Github, this library has been used all along this thesis to perform benchmarks and experiments.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France