SNA course project

# Simulating rare events Hawkes process applied to *Twitter*

Authors

Y. El Maazouz & M. A. Bennouna

Supervisors

| | |
|---|---|
| I. Kortchemski | F. Benaych-Georges |
| S. De Marco | M. Bompaire |
| E. Gobet | G. Fort |

04 June 2018

# Contents

**Abstract**

We present in this article a simulation of messages in *Twitter* (tweets) using Hawkes process. A tweet is modeled by its tweeting time and features (tags characterizing the tweet's content) . We consider in a first approach an optimized Thinning algorithm and in a second one Ogata's algorithm.

We show some theoretical results on the evolution of the number of tweets that follows from our model. We also prove similar results for the model with features.

With the tools constructed, we proceed to a simulation of rare events (probability below $10^{-10}$) on the evolution of the number of messages through time and we deduce probabilities of events like a *tweet-apocalypse*.

Finally, to make our work applicable for real data, we present a maximum likelihood estimator of parameters of our model to fit to real life evolution.

## 1 Simulating Hawkes process

### 1.1 Simulating with a superposition of Poisson process

#### 1.1.1 Thinning simple algorithm

A Hawkes point process [3] is a self-excitation point process $(N_t)_{t\geq 0}$ that has a density $\lambda(t) = \lambda_0 + \sum g(t-t_i)1_{t_i<t}$ where $(t_i)_{i\geq 1}$ are jump times of $(N_t)_{t\geq 0}$ and $g$ a decreasing function on $\mathbb{R}^+$

A first approach to simulate such a process in an interval $[0,T]$, with $T$ a fixed time horizon, is to simulate a superposition of several Poisson processes as follow:

We generate a first generation of jump times $\{t_i^{(0)} : i \geq 1\}$ as an independent Poisson process of intensity $\lambda_0$

$$\Pi_0 = \{t_i^0 : i \geq 1\} \sim PP(\lambda_0)$$

We generate the k-th generation of jump times $\{t_i^k : i \geq 1\}$ as an independent Poisson process of intensity $\sum g(t-t_i^{(k-1)})1_{t_i^{(k-1)}<t}$ :

$$\Pi_k = \{t_i^k : i \geq 1\} \sim PP(\sum g(t-t_i^{(k-1)}).1_{t_i^{(k-1)}<t})$$

The Hawkes process correspond to a superposition of $(\Pi_i)_{i\geq 1}$ which is:

$$\Pi = \cup_{i\geq 1}\Pi_i$$

In order to do that, we need to simulate inhomogeneous Poisson processes (i.e with variable intensity in time). We chose to use in a first approach Thinning algorithm.

Thinning algorithm consist of finding an upper bound $\bar{\lambda}$ of the density $\lambda$, simulating jump times with an inhomogenious Poisson process $PP(\bar{\lambda} * T)$ and to keep every jump time $t_i$ with probability $\frac{\lambda(t_i)}{\bar{\lambda}}$.

The following figures illustrates some simulation results.

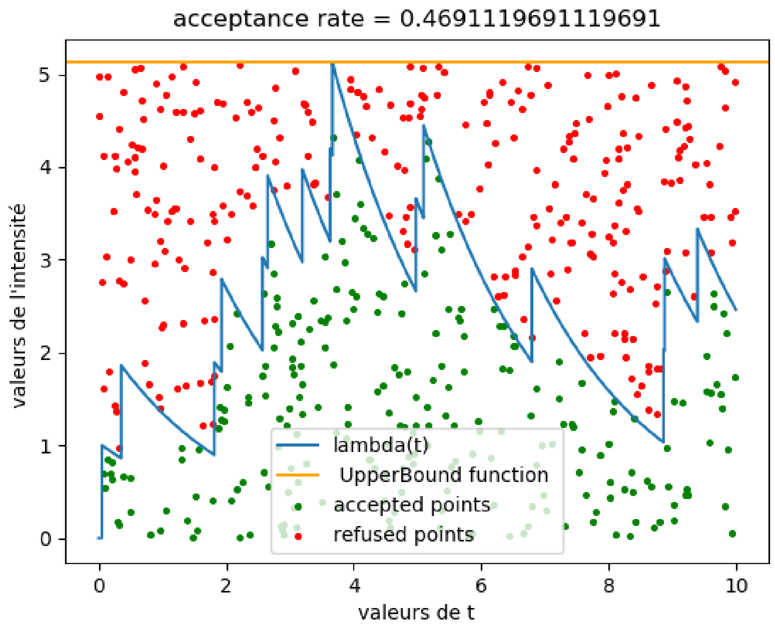


Figure 1: Thinning algorithm with $\bar{\lambda} = \max \lambda(t)$

**For an excitation function:** $g(t) = \alpha e^{-\beta t} 1_{t \geq 0}$ **:**
(figure for $T = 100.$, $\lambda_0 = 1$ ,$\alpha = 1$ et $\beta = 1$, simulation cost: 0.651 seconds)

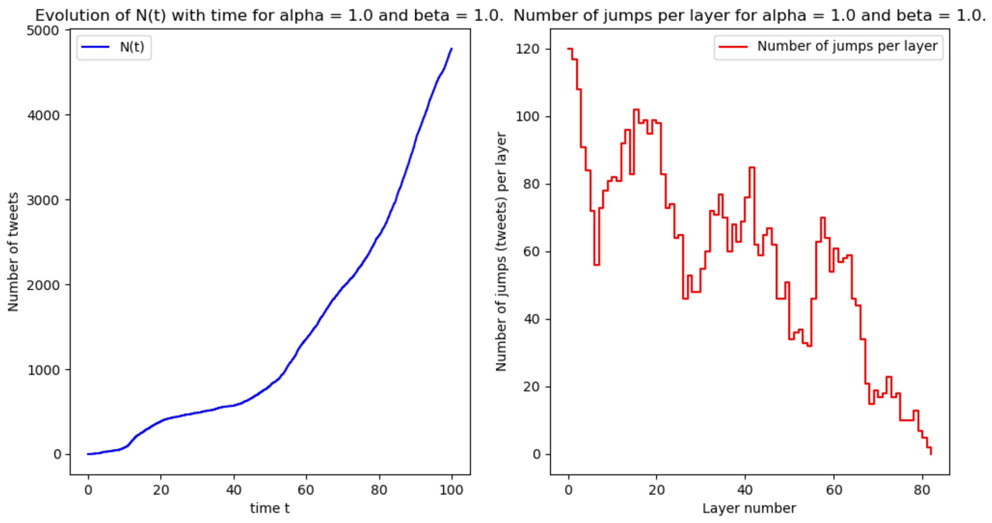


Figure 2: Simulating Hawkes processes

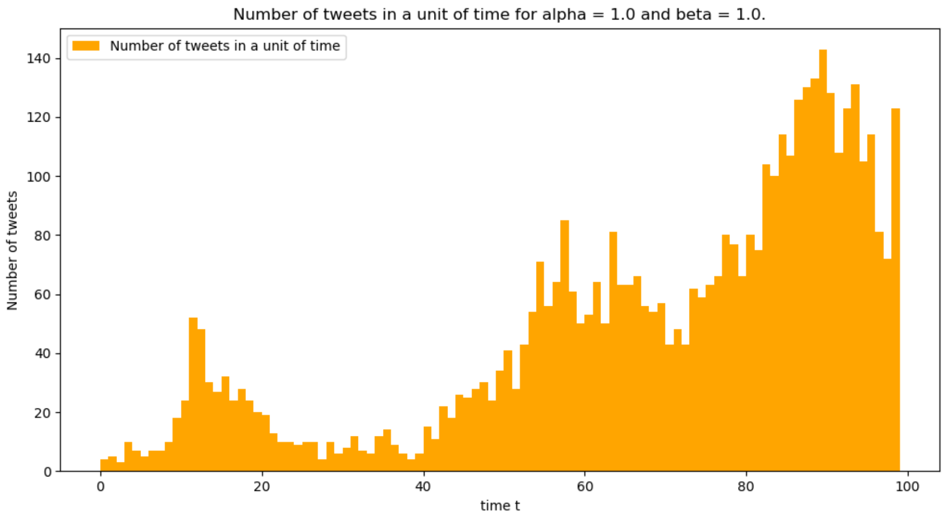


Figure 3: Number of tweets per second

### 1.1.2 Improving the algorithm

One of the problems we face with this simulation method is the simulation time that increase fast for big values of $T$ and for some sets of $\alpha$ and $\beta$. We would like to be able to simulate Hawkes process for a day duration (86400 seconds), we need therefor to improve the algorithm.

Is this first algorithm, the acceptance rate of the generated points is very low as shows the figure 1. This is caused by the choice of $\bar{\lambda}$ that is far from being optimal.

In order to improve the acceptance rate and thus to reduce the simulation time, we chose $\bar{\lambda}$ as a step function (figure 4). With this method, the acceptance rate is much higher (0.77 now and 0.47 before) and the algorithm is faster.

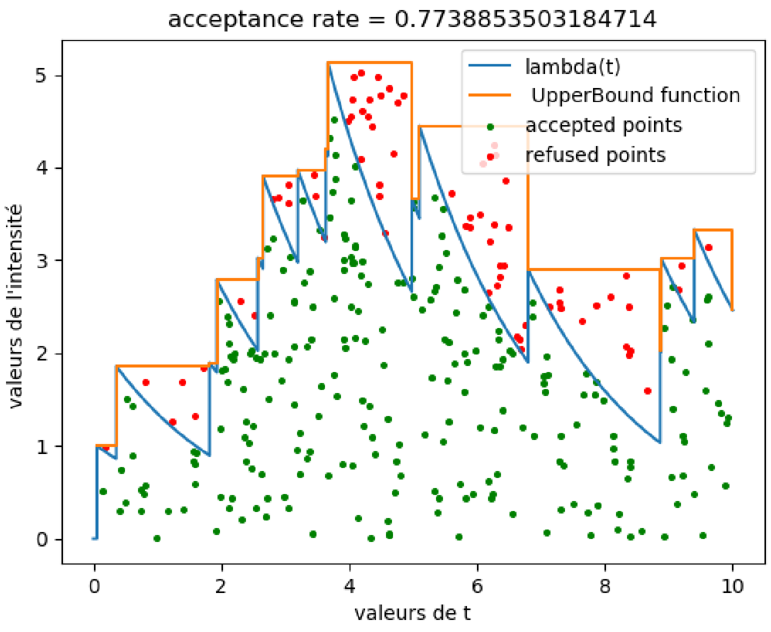


Figure 4: Thinning sample algorithm with $\bar{\lambda}$ as a step function

($T = 100.$, $\lambda_0 = 1$ ,$\alpha = 1$ et $\beta = 1$, Simulation time: 0.224 seconds)

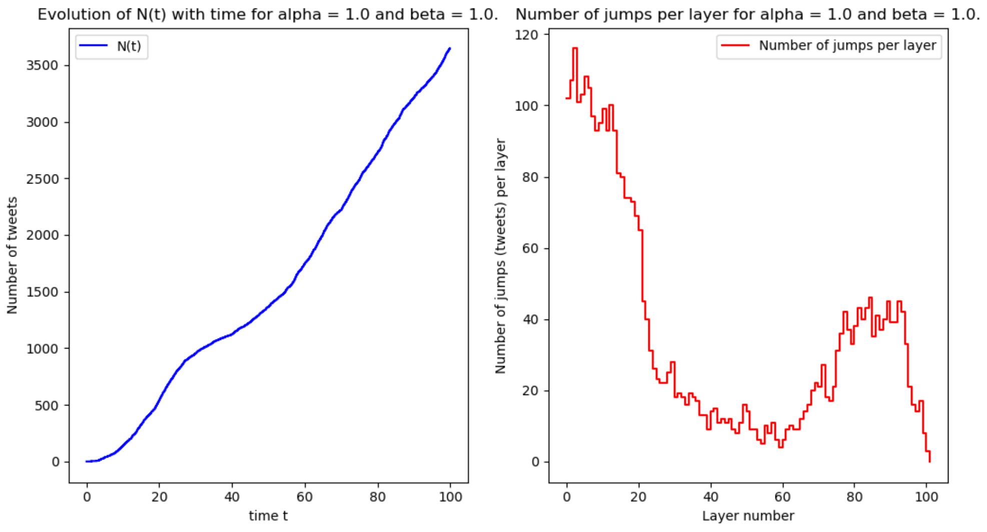


Figure 5: Hawkes process simulation with $\bar{\lambda}$ as a step function

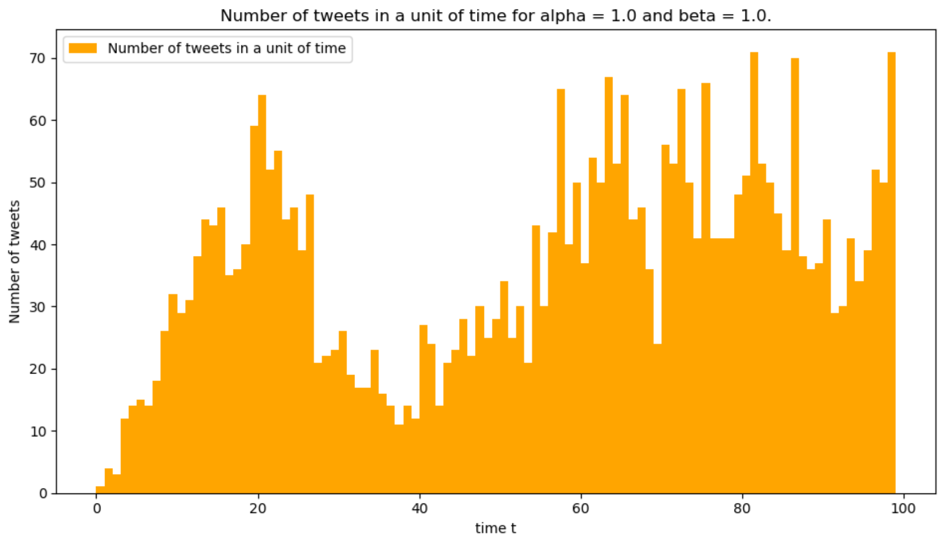


Figure 6: Number of tweets per second.

## 1.2 Ogata Algorithm

In spite of $\bar{\lambda}$ improvement, the previous algorithm takes too much time for values of T around a day.

We explored other methods in literature and chose to implement Ogata algorithm [2].

Ogata algorithm is considerably faster than Thining algorithm. The choice of $\bar{\lambda}$ is optimized in every new point's simulation, the acceptance rate is therefore higher. Avoiding the layer simulation improves the speed as well.

Another considerable optimization is the use of the exponential propriety. We can compute more efficiently $\lambda(s)$ with the recurrence relation: $\lambda(s) = \lambda_0 + e^{-\beta w}(\bar{\lambda} - \lambda_0)$ and $\bar{\lambda} = \lambda(s) + \alpha 1_{s=t_{i+1}}$, where $t_i$ is the last selected jump time. This optimization is only possible when using an excitation function in exponential form.

All the results that follow in this article are obtained by using Ogata algorithm.

**Algorithm 1** Ogata algorithm for excitation $g(x) = \alpha e^{-\beta x}$ in $[0, T]$

```
Input: $\alpha$, $\beta$, $T$, $\lambda_0$
Initialize JUMPTIMES := $\mathbb{JT} = \emptyset$, CURRENTINSTANT:= $s = 0$, $n = 0$;
while s<T do do
    Set $\bar{\lambda} = \lambda(s^+) = \lambda_0 + \sum_{\tau \in \mathbb{JT}} \alpha e^{-\beta(s-\tau)}$
    Generate jumpGapWidth:= $w \sim \mathcal{E}(\bar{\lambda})$;
    $s = s + w$;                    # Next candidate
    Generate $D \sim$ uniform(0,1);
    if D $\bar{\lambda} \leq \lambda(s) = \lambda_0 + \sum_{\tau \in \mathbb{JT}} \alpha e^{-\beta(s-\tau)}$ then        # Accept with probability $\lambda(s)/\bar{\lambda}$
        $n = n + 1$;                    # Update number of points
        $t_n = s$;
        $\mathbb{JT} = \mathbb{JT} \cup \{t_n\}$;                # Add the n-th point to JUMPTIMES
    end if
end while
if $t_n \leq T$ then
    return $\{t_k\}_{k=1,2,\ldots,n}$
else
    return $\{t_k\}_{k=1,2,\ldots,n-1}$
end if
```

## 1.3 Numba and C++

In order to boost our program's performances, we have rewritten our code in C++ and compared the performances to those of python. Naturally the C++ code executed way faster than python but it was not as fast as we hoped for.

Our supervisors then advised us to use a module in python called numba to accelerate python code execution.

When coding is Python, using the decorator `jit` from the library `Numba` it translates Python functions to optimized machine code at runtime. This method proved to be faster than our C++ code.

## 2 Theoretical results

To chose the right simulation parameters in our work, we need theoretical results on the behaviour of $N_T$.

### 2.1 Different behaviours

We noticed during the prior simulations two different behaviours of $N_t$ depending on the values of $\alpha$ and $\beta$. This behavior splits into to two different cases: *defective* case ($\frac{\alpha}{\beta} < 1$) and *excessive* case ($\frac{\alpha}{\beta} > 1$) with a *critical* case ($\frac{\alpha}{\beta} = 1$) in between.

#### 2.1.1 *defective* case :$\frac{\alpha}{\beta} < 1$

The expected value of $N_T$ is finite and we have the following results [4]:

$$\mathbb{E}(\lambda(t)) = \lambda_\infty + (\lambda_0 - \lambda_\infty)(1 - e^{-(\beta-\alpha)t})$$

$$\mathbb{E}(N_t) = \lambda_\infty t + \frac{\lambda_0 - \lambda_\infty}{\beta - \alpha}(1 - e^{-(\beta-\alpha)t})$$

where

$$\boxed{\lambda_\infty = \frac{\beta\lambda_0}{\beta - \alpha} = \frac{\lambda_0}{1 - \frac{\alpha}{\beta}}}$$

For big values of $T$ :

$$\mathbb{E}(\lambda(T)) \xrightarrow{T\to\infty} \lambda_\infty$$

$$\boxed{\mathbb{E}(N_T) \xrightarrow{T\to\infty} \lambda_\infty T = \frac{\lambda_0}{1-\frac{\alpha}{\beta}} T} \tag{1}$$

$$\boxed{\mathrm{Var}(N_T) \xrightarrow{T\to\infty} \frac{\beta^2 \lambda_\infty}{(\beta-\alpha)^2} T = \frac{\lambda_0}{(1-\frac{\alpha}{\beta})^3} T}$$

The proof of this result is similar to the proof presented section 4.2.

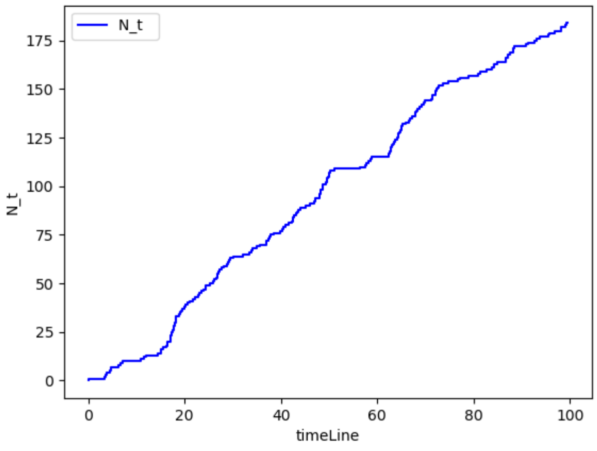


Figure 7: Simulation of the *defective* case $\frac{\alpha}{\beta} < 1$, $T = 100$

### 2.1.2 *excessive* case: $\frac{\alpha}{\beta} > 1$

The process diverges exponentially fast [4]:

$$\boxed{\lambda(t) \to \infty}$$

$$\boxed{N_t \to \infty}$$

For a set of parameters with $\frac{\alpha}{\beta} > 1$, the simulation diverges fast or don't have time to end. The values that the first algorithm gives for each layer were increasing fast in the earliest generations. and the values of $N_T$ are much bigger than the *defective* case.

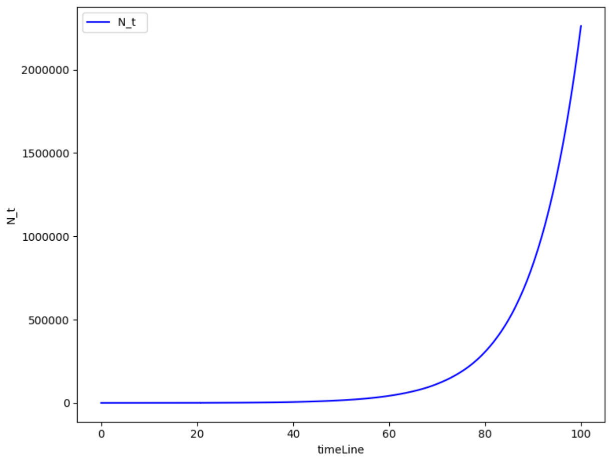
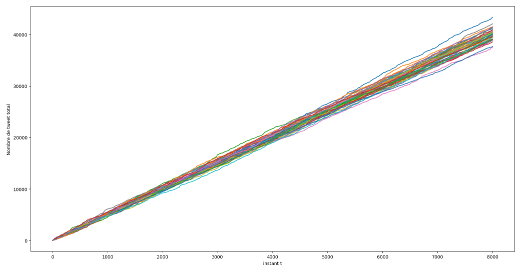


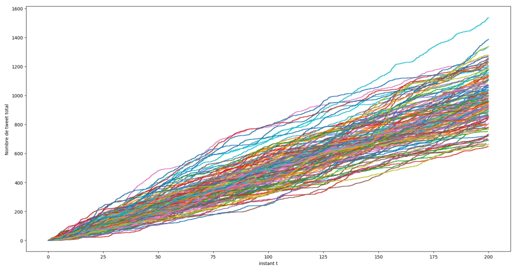
Figure 8: Simulation of the *excessive* case $\frac{\alpha}{\beta} > 1$, $T = 100$

## 2.2 On the values of the expect value and the variance for the *defective* case

We notice that $\sigma(N_t) \propto \sqrt{T}$ and $\mathbb{E}(N_t) \propto T$. Such a result can be expected given the simulations. The evolution of $N_T$ for different scales of $T$ suggest this proportionality ($\frac{\sigma(N_t)}{\mathbb{E}(N_t)} \propto \frac{1}{\sqrt{T}}$) :



$T = 8000$



$T = 200$

Figure 9: Comparing the spreading of trajectories for two values of $T$

## 3 Self excitation with features

### 3.1 Model

Tweets contain features (words, symbols, URLs ...). These features influence the number of tweets caused directly or indirectly by the original tweet. To take feature's effect into account, we modified Ogata code to change the way previous tweets influence future tweets generation.

Every tweet has now a feature selected from a finite set $\mathcal{F}$ of features. In a first simple model, a feature is attributed randomly to a generated tweet with a probability distribution $\pi$ on $\mathcal{F}$.

To model the influence of these features, we modified the excitation, as suggested in [5], by multiplying the excitation function by a coefficient depending on the feature of the tweet. A tweet having a feature $f \in \mathcal{F}$ has therefore the following excitation function: $g(t, f) = \gamma(f)\alpha \exp(-\beta t)$ where $\gamma(f)$ is the coefficient specific to the feature $f$.

Thus, the excitation intensity with features take the following form:

$$\lambda(t) = \lambda_0 + \sum_{t_i \leq t} \gamma(f_{t_i})\alpha e^{-\beta(t-t_i)}$$

where $f_{t_i}$ is the feature corresponding to the tweet at time $t_i$.

### 3.2 Expected value and variance of the process with features

The form of our initial model suggest and intuitive expression of the expected value and the variance of $N_T$:

$$\mathbb{E}(N_T) = \frac{\lambda_0}{1 - \mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta}} T$$

$$\mathrm{Var}(N_T) = \frac{\lambda_0}{(1 - \mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta})^3} T$$

The numerical results are close to these values. The proof of the initial model can be adapted to the one with features.

**Proposition 3.1** *For a Hawkes process with features,* $\mathbb{E}(N_T)$ *depend on the value of* $n = \int_0^\infty \mathbb{E}_\pi(\gamma)g(s)ds = \mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta}$.
*if* $n > 1,\ \ \mathbb{E}(N_T) \to \infty$
*if* $n < 1,\ \ \mathbb{E}(N_T) = \frac{\lambda_0}{1-\mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta}} T$

*Proof*
The proof is inspired by the case of 1.

$$e(t) := \mathbb{E}(\lambda(t)) = \mathbb{E}\Big(\lambda_0 + \int_0^t \gamma(f_s)g(t-s)dN(s)\Big)$$
$$= \lambda_0 + \int_0^t g(t-s)\mathbb{E}(\gamma(f_s)dN(s))$$

$\gamma_s$ and $dN(s)$ are independent is our model, therefore:

$$e(t) = \lambda_0 + \int_0^t g(t-s)\mathbb{E}_\pi(\gamma)\mathbb{E}(dN(s)) \tag{2}$$

Noting $\mathcal{F}(s)$the knowledge of the events before time $s$,

$$\lambda(s) = \frac{\mathbb{E}(dN(s)|\mathcal{F}(s))}{ds}$$

By taking the expected value of this expression,

$$e(t) = \mathbb{E}\Big(\frac{\mathbb{E}(dN(s)|\mathcal{F}(s))}{ds}\Big) = \frac{\mathbb{E}(dN(s))}{ds}$$

We inject this result in the expression 2 and we obtain:

$$e(t) = \lambda_0 + \int_0^t g(t-s)\mathbb{E}_\pi(\gamma)e(s)ds = \lambda_0 + \int_0^t e(t-s)\mathbb{E}_\pi(\gamma)g(s)ds$$

Thus, $e(t)$ obey to a convolution equation $e = \lambda_0 + e * \mathbb{E}_\pi(\gamma)g$. This equation's solution depends on the value of $n = \int_0^\infty \mathbb{E}_\pi(\gamma)g(s)ds = \mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta}$ [4].

**_excessive_ case:** $\mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta} > 1$

The number of tweets diverges exponentially. We could expect this result given the model without features: the number of tweets diverges when $\frac{\alpha}{\beta} > 1$.

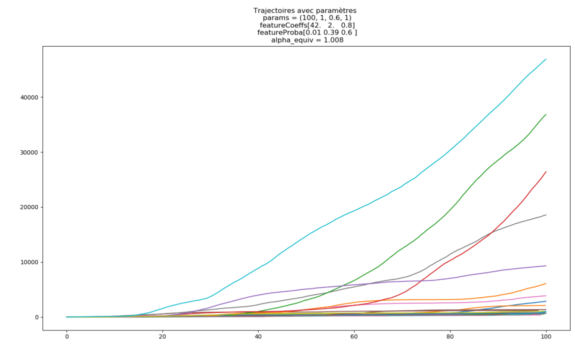


Figure 10: Simulation with features $\mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta} > 1$

## ***defective* case:** $\mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta} < 1$

The number of tweets expected value is finite: $\mathbb{E}(N_T) = \frac{\lambda_0}{1-\mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta}}T$ This expression is similar to the one without features with $\alpha \leftarrow \alpha\mathbb{E}_\pi(\gamma)$.

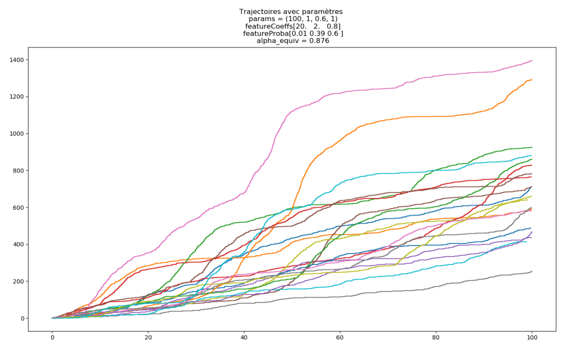


Figure 11: Simulation with features $\mathbb{E}_\pi(\gamma)\frac{\alpha}{\beta} < 1$

### 3.3 Improving the model

We can modify our simple features model to better capture the real impact of features of the tweets. A possibility would be that the distribution of features allocation depends on the past with a function $g$:

$$\pi(f_{n+1}|f_1, .., f_n, t_1, ..t_{n+1}) = g(f_{n+1}, ..., f_1, t_1, ..t_{n+1})$$

## 4 Estimation of rare events probability

### 4.1 Estimation method

We wish to estimate the probability $P(N_T > a)$ for *high values* of $a$ (to estimate the probability of a tweet-apocalypse in a period of time T for example).

We have chosen importance sampling methods in order to accelerate the convergence of our estimator and because such methods are easier to implement. For this we are using the likelyhood in Ogata's article [2], where $\mathbb{P}_0$ represents the probability distribution of a standard Poisson process:

$$\frac{d\mathbb{P}}{d\mathbb{P}_0} = \exp\Big(\int_0^T \log\lambda(t)dN_t + \int_0^T (1-\lambda(t))dt\Big)$$

For a set of parameters $\theta_1 = (T, \lambda_{0,1}, \alpha_1, \beta_1)$ we denote by $\mathbb{P}_1$ the probability distribution associated to the Hawkes process with parameter $\theta_1$. To estimate the probability $\mathbb{P}_1(N_T > a)$ we operate a probability change $\mathbb{P}_1 \to \mathbb{P}_2$ for a parameter $\theta_2 = (T, \lambda_{0,2}, \alpha_2, \beta_2)$, the choice of the parameter $\theta_2$ will be discussed later on.

Ogata's article gives the likelihood of this probability change:

$$\frac{d\mathbb{P}_1}{d\mathbb{P}_2} = \frac{d\mathbb{P}_1}{d\mathbb{P}_0}.\frac{d\mathbb{P}_0}{d\mathbb{P}_2} = \frac{d\mathbb{P}_1}{d\mathbb{P}_0}\Big/\frac{d\mathbb{P}_2}{d\mathbb{P}_0}$$

$$\frac{d\mathbb{P}_1}{d\mathbb{P}_2} = \exp\Big(\int_0^T \log\frac{\lambda_1(t)}{\lambda_2(t)}dN_t + \int_0^T (\lambda_2(t)-\lambda_1(t))dt\Big)$$

For our model, the log-likelihood is:

$$\log\Big(\frac{d\mathbb{P}_1}{d\mathbb{P}_2}\Big) = \mathcal{L}(T, \lambda_{0,1}, \alpha_1, \beta_1) - \mathcal{L}(T, \lambda_{0,2}, \alpha_2, \beta_2)$$

where:

$$\mathcal{L}(T, \lambda_0, \alpha, \beta) = \sum_{i=1}^{N_T}\log(\lambda_i) + T(1-\lambda_0) - \sum_{i=1}^{N_T}\frac{\alpha}{\beta}(1-e^{-\beta(T-t_i)})$$

We can then estimate the probability by simulating Hawkes processes with the parameter

$$\theta_2$$

, ie:

$$\mathbb{E}_{\mathbb{P}_1}\big(1_{N_T>a}\big) = \mathbb{E}_{\mathbb{P}_2}\big(1_{N_T>a}\frac{d\mathbb{P}_1}{d\mathbb{P}_2}\big)$$

The law of large numbers then allows us to build two estimators for the probability $\mathbb{P}_1(N_T \geq a)$.

Let $L$ be the likelihoods $d\mathbb{P}_1/d\mathbb{P}_2$ and $M$ an integer $\geq 1$, And let $(N_T^{(k)})_{1\leq k\leq M}$ $(V^{(k)})_{1\leq k\leq M}$ be independent realizations (or copies) of $(N_T)$ and $V$ we can then build two estimator of $\mathbb{P}_1(N_T \geq a)$ by considering:

$$u_M = \frac{1}{M}\sum_{k=1}^{M} 1_{N_T^{(k)}\geq a}$$

$$v_M = \frac{1}{M}\sum_{k=1}^{M} 1_{N_T^{(k)}\geq a} V^{(k)}$$

where the instants $t_i$ are simulated with probability distribution $\mathbb{P}_1$ for the estimator $u_M$ (called the naive Monte-Carlo estimator) and with probability distribution $\mathbb{P}_2$ for the estimator $v_M$ (called the importance sampling estimator).

## 4.2 Confidence intervals:

Let's consider: $\mu := \mathbb{P}_1(N_T \geq a)$, $\sigma_u = \sqrt{\mathrm{Var}_1(1_{N_T\geq a})}$ et $\sigma_v = \sqrt{\mathrm{Var}_2(1_{N_T\geq a}V)}$.

With the Centrale Limite theorem:

$$\frac{\sqrt{M}}{\sigma_u}(u_M - \mu) \xrightarrow[M\to+\infty]{Loi} \mathcal{N}(0,1)$$

$$\frac{\sqrt{M}}{\sigma_v}(v_M - \mu) \xrightarrow[M\to+\infty]{Loi} \mathcal{N}(0,1)$$

Then, if we can calculate the two variances ${\sigma_u}^2$ and ${\sigma_v}^2$ we can get asymptotic confidence intervals for our estimators

However, we do not have an exact formula that allows us to calculate these variances, and by the way if such a formula existed we would have been able to calculate $\mu$ from from the expression of ${\sigma_u}^2$

To overcome this difficulty we can replace the two standards deviations $\sigma_u$ and $\sigma_v$ with their respective estimators $\sigma_{u,M}$ and $\sigma_{v,M}$ where:

$$\sigma_{u,M}^2 = \frac{1}{M}\sum_{k=1}^{M} 1_{N_T^{(k)}} - \big(\frac{1}{M}\sum_{k=1}^{M} 1_{N_T^{(k)}}\big)^2$$

$$\sigma_{v,M}^2 = \frac{1}{M}\sum_{k=1}^{M} (1_{N_T^{(k)}} V^{(k)})^2 - \big(\frac{1}{M}\sum_{k=1}^{M} 1_{N_T^{(k)}} V^{(k)}\big)^2$$

Let $q_{1-\alpha/2}$ be the quantile of order $1-\alpha/2$ of a standard Normal distribution. We then write the two asymptotic confidence interval as follows:

$$I_{u,M} = [u_M - \frac{q_{1-\alpha/2}}{\sqrt{M}}\sigma_{u,M} ; u_M + \frac{q_{1-\alpha/2}}{\sqrt{M}}\sigma_{u,M}]$$

$$I_{v,M} = [v_M - \frac{q_{1-\alpha/2}}{\sqrt{M}}\sigma_{v,M} ; v_M + \frac{q_{1-\alpha/2}}{\sqrt{M}}\sigma_{v,M}]$$

## 4.3 Primary tests

The primary calculations of the likelihood almost always gave us 0.0. We had to explore many possibilities to understand the source of this problem. It was either an implementation error of a bad choice of parameters. And at first we had no quantification of the likelihood and we had no idea of what to expect as a result by changing the parameters.

As a first test we verified that the following condition is satisfied by our simulations:

$$\mathbb{E}_{\mathbb{P}_2}\left(\frac{d\mathbb{P}_1}{d\mathbb{P}_2}\right) = 1 \tag{3}$$

This test helped discover some errors in our code and gave us an idea on the performance of our implementation.

After making the necessary correction to our code the results when estimating $\mathbb{E}_{\mathbb{P}_2}\left(\frac{d\mathbb{P}_1}{d\mathbb{P}_2}\right)$ were reasonable (between 0.5 and 1.5 with $10^4$ simulations)

The fact that the results are not always close to 1 is normal. Actually we are simulating a random variable that is of type $(t_1, t_2, ..., t_{N_T})$ with $\mathbb{E}(N_T) \sim 400$. It is then comprehensible that our naive Monte Carlo estimator of $\mathbb{E}_{\mathbb{P}_2}\left(\frac{d\mathbb{P}_1}{d\mathbb{P}_2}\right)$ does not perform well with $10^4$ because we need a number of simulations of around $n \times N_T$ to get good estimation results.After we used `Numba` in python to accelerate the execution of our code we have been able to get satisfying results for this condition.

Let $\theta_1 = (T, \lambda_{0,1}, \alpha_1, \beta_1)$ be a set of parameters, and $\mu_1 = \mathbb{E}_{\mathbb{P}_1}(N_T)$ et $\sigma_1^2 = \text{Var}_{\mathbb{P}_1}(N_T)$ (for which we have a formula). Remember that the objective is to estimate the probabilities of events $\mathbb{P}(X > a)$ where $a$ is given ($a$ is a large number for rare events) with using a probability change to accelerate the convergence of the estimator. We must the choose the right parameter $\theta_2 = (T, \lambda_{0,2}, \alpha_2, \beta_2)$ for the probability change in order to get a good acceleration.

By simulating a sample of $N_T$ and plotting a histogram of it's values, we have noticed that it's distribution is close to a normal distribution

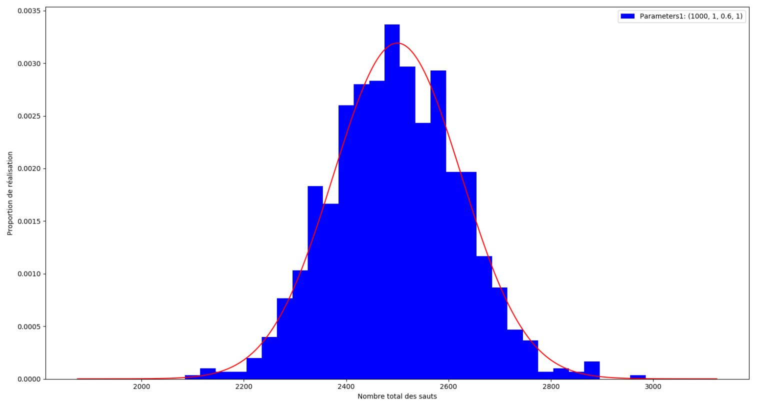


Figure 12: Approximating the distribution of $N_T$ with a Gaussian.

To have an idea of the probability value to expect, we chose the value of $a$ as a quantile of a $\mathcal{N}(\mu_1, \sigma_1^2)$ of order $1 - 10^{-m}$ (the value of probability to excpect then is of order $10^{-m}$)

For the parameter $\theta_2 = (T, \lambda_{0,2}, \alpha_2, \beta_2)$ we chose the following settings:

1. $\mu_2 = a$: This choice allows us to center the distribution $\mathbb{P}_2$ arround $a$ to have a 50-50 chance of going over the barrier $a$ with the second probability distribution $\mathbb{P}_2$.

2. $\alpha_2 = \alpha_1$, $\beta_2 = \beta_1$: We have noticed that by choosing a parameter setting the changes the ratio : $\frac{\alpha}{\beta}$ the likelihood is always too small and the effects on $\mathbb{E}(N_T)$ are considerable when compared to changing only the parameter $\bar{\lambda}_0$. The reason behind that is that by changing that ratio the nature of trajectories changes as well in such a way that the probability distribution $\mathbb{P}_2$ does not resemble at all to $\mathbb{P}_1$ because the likelihood depends on the entire trajectory of

instants $t_i$ and not just the final value $N_T$ ie: by changing $\alpha$ for example we change the entire form of $\lambda(t)$. Using the result of section 3 we chose: $\lambda_{0,2} = \frac{\mu_2}{T}(1 - \frac{\alpha_1}{\beta_1})$.

( We have also tried to operate the probability change by changing the parameter $\alpha$ and adapting $\mu_2$ et $\sigma_2$ but that proved to be less efficient. )

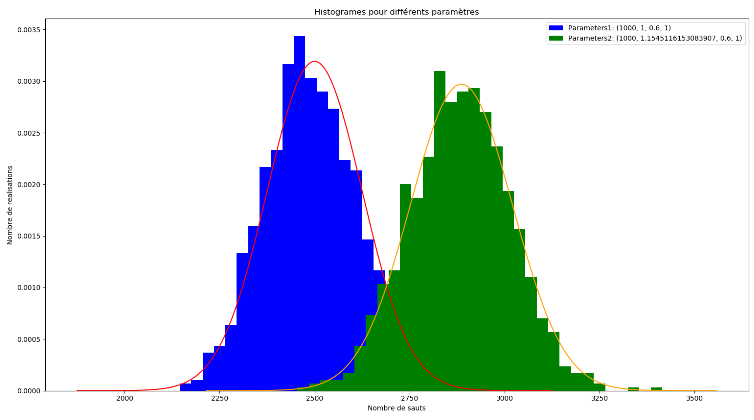


Figure 13: Probability change by centring.

Our code is also compatible with the model with features. the results obtained in this case were similar.

In our implementation we also took into consideration the tweet-fatures. And the estimator was equally efficient.

### 4.4 Examples:

Let $\theta_1 = (T = 1000, \lambda_{0,1} = 1, \alpha_1 = 0.3, \beta_1 = 1)$.

We recall the results obtained in section 3 for the expectation and variance of $N_T$ (with $(N_t)$ generated under the probability distribution $P_{\theta_1}$) for large values of $T$, $T \to +\infty$:

$$\mathbb{E}(N_T) \simeq \frac{\lambda_{0,1}}{1 - \frac{\alpha_1}{\beta_1}} T$$

$$\mathrm{Var}(N_t) \simeq \frac{\lambda_{0,1}}{(1 - \frac{\alpha_1}{\beta_1})^3} T$$

In the following examples we estimate the probability $\mathbb{P}_{\theta_1}(N_T > a)$ where $a$ is chosen to be the quantile of order $10^{-m}$ of the normal distribution approximating that of $N_T$.

<u>Importance sampling when changing $\lambda_0$ :</u>

Remark:

The following results were obtained with $M \times n$ estimations of $N_T$ to build the confidence intervals. As for the boxPlots, we represented n values of our estimators where each estimator value is computed with $M$ simulations of $N_T$.

For $m = 2$ (probability of order $10^{-2}$ ):
Estimation results for $M = 100$ and $n = 100$:
MonteCarlo Naïf:
  Estimated Value: 0.0103
  Confidence interval with level 95% $[0.00832; 0.01227]$
  Relative Error: 0.3842

Importance Sampling:
    Estimated Value: 0.01074
    Confidence interval with level 95% $[0.01034; 0.01074]$
    Relative Error: 0.02356

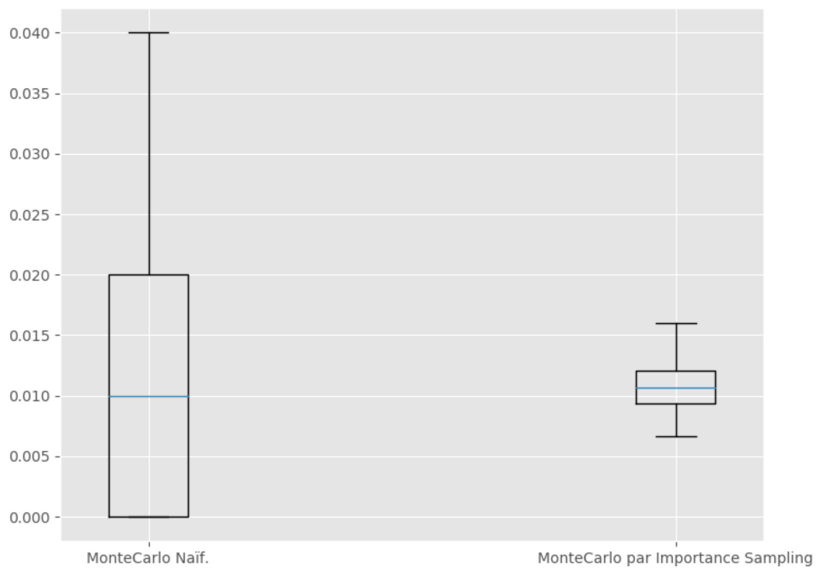


Figure 14: BoxPlot pour n = 100 valeurs des deux estimateurs $u_M$ et $v_M$.

Notice that even for a proibability of order $10^{-2}$, which is not considered to be a rare event, the Naive Monte-Carlo estimator is less efficient compared to the importance sampling estimator (less relative error and narrower confidence interval).

We can enhance the performance of the estimator by simply increasing simulations number $M$

For M = 1000 and n = 100, we get the following results:

MonteCarlo Naïf:
    Estimated Value: 0.01102
    Confidence interval with level 95% $[0.01037; 0.11743]$
    Erreur Ralative: 0.11743

Importance Sampling:
    Estimated Value: 0.01058
    Confidence interval with level 95% $[0.010455; 0.010704]$
    Relative Error: 0.02356

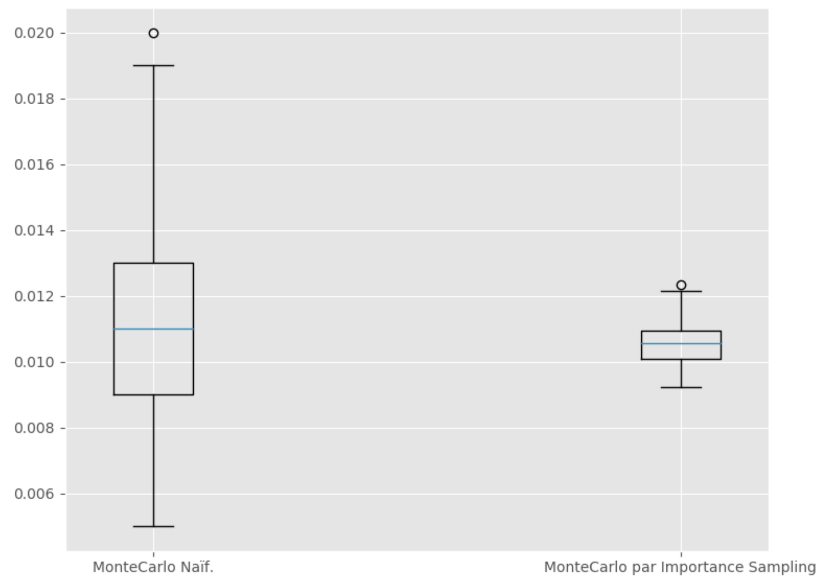


Figure 15: BoxPlot for n = 100 values of two $u_M$ et $v_M$.

For $m = 2$ (probability of order $10^{-2}$ ): (this is where the importance sampling method is way more interesting)

for M=1000 and n = 100:

MonteCarlo Naïf:

Estimated Value: $0, 0$

Confidence interval with level 95% $[0.0; 0.0]$

Erreur Ralative: Nan

Importance Sampling:

Estimated Value: $3.253 \times 10^{-7}$

Confidence interval with level 95% $[3.167 \times 10^{-7}; 3.758 \times 10^{-7}]$

Relative Error: 0.0527

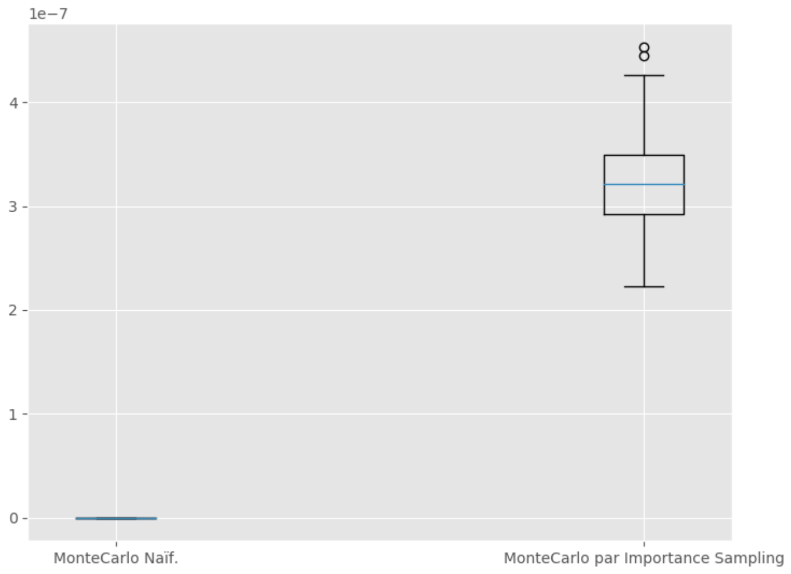


Figure 16: Algorithme de choc fictif.

Notice that the naive Monte-Carlo estimator does not converge quickly enough to give us a meaningful estimation of the probability we are estimating (which is too small: $10^-7$). This is because we are simulating only $M = 1000 = 10^(3)$ times the variable $N_T$ when we do need around $10^7$ simulations to reasonable results which is a lot. Importance sampling estimator allows a good estimation of the probability value with a median of around $3,3.10^{-7}$ and a relative error of 0.05. This estimator already gives good results for only $10^3$ simulations.

Importance sampling by changing $\alpha$ :

We can also operate a probability change by changing $\alpha$ of the process to have a distribution centered on the threshold $a$ (Here, the threshold is a quantile of a Gaussian distribution that approaches or initial distribution). The form on $N_T$ variance shows that it's increasing in $\alpha$. We can therefore change the parameters $\alpha$ and $\lambda_0$ to have a distribution centered on $a$ and a small variance as shown in the following figure:

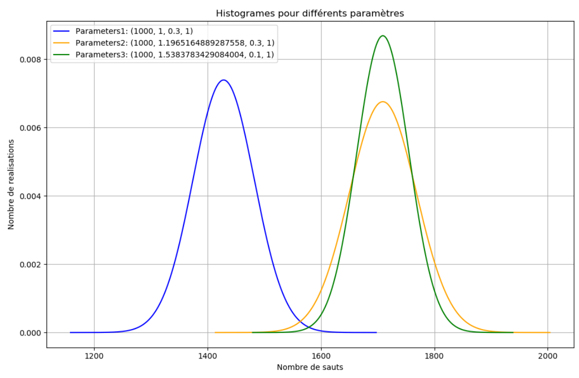


Figure 17: Probability changing.

By reducing $N_T$ variance, we expect to reduce the variance of $N_T 1_{N_T > a}$ and therefore the confidence interval. We will first attempt to do it by changing $\alpha$ only.

For the set of parameters $\theta_1$ considered, and to estimate a probability of order $10^{-7}$, we have the following result with changing only $\alpha$:

For $M = 1000$ and $n = 100$:

Naive MonteCarlo:

Estimated Values: 0.0

$[0.0; 0.0]$

Relative Error: nan

Importance Sampling:

Estimated Values: $3.1452 \times 10^{-7}$

Confidence interval with level 95% $[2.88294 \times 10^{-7}; 3.40765 \times 10^{-7}]$

Relative Error: 0.16

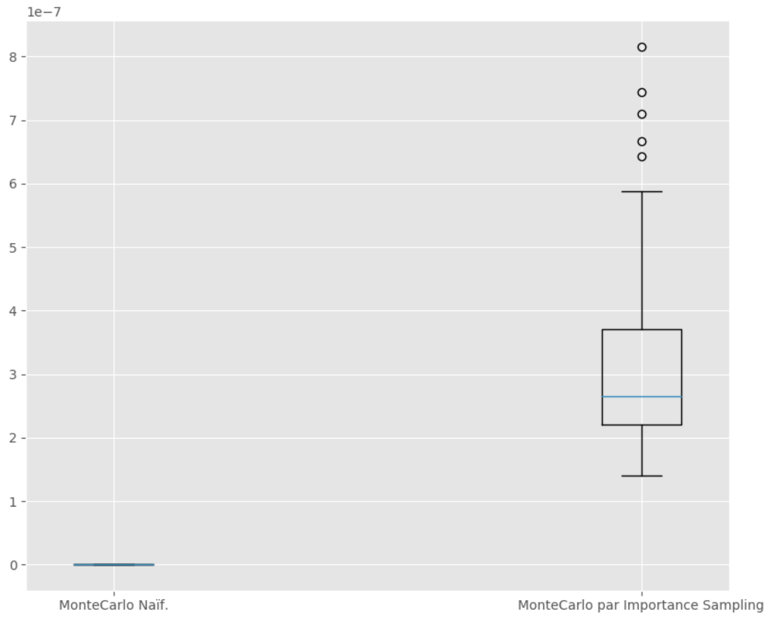


Figure 18: BoxPlot for n = 100 values with two estimators $u_M$ and $v_M$.

We notice that the values in this case are more scattered compared to the case where we only change $\lambda_0$ (relative error : 0.16 compared to 0.052

This result can be expected as the variance of $N_T$ is bigger in the case of changing $\alpha$. The dependency of $Var(N_T)$ in $\alpha$ is as $\alpha^3$.

For the set of parameters $\theta = (86400, 1, 0.3, 1)$, and a threshold $a = 127268$ and $M = 10000$.

Naive MonteCarlo:

Estimated Values: 0.0

Confidence interval with level 95% $[0.0; 0.0]$

Relative Error: nan

Importance Sampling:

Estimated Values: $1.6831 \times 10^{-14}$

Confidence interval with level 95% $[1.57849 \times 10^{-14}; 1.78783 \times 10^{-14}]$

Relative Error: 0.12

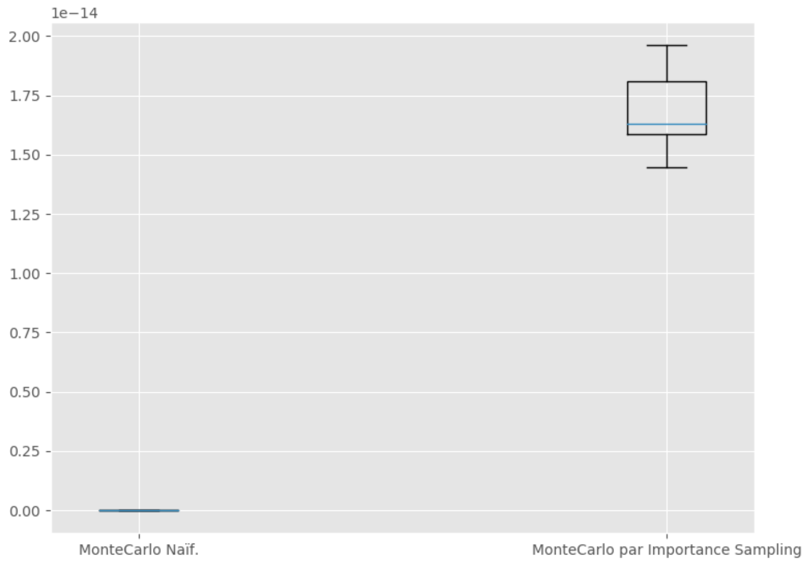


Figure 19: BoxPlot for n = 10 values with two estimators $u_M$ and $v_M$.

## 5 Estimating parameters by likelihood maximisation

To make our results usable in real life, we need to estimate real values of $\lambda_0$, $\alpha$ and $\beta$. In oder to do that, we used a maximum likelihood estimator $\widehat{\lambda_0}_n, \widehat{\alpha}_n, \widehat{\beta}_n$.For real data $(t_i)_{1\leq i\leq n}$ respresenting tweets times in a day, we want to maximize the log-likelihood:

$$\mathcal{L}(T,\lambda_0,\alpha,\beta) = \sum_{i=1}^{n} \log(\lambda_i) + T(1-\lambda_0) - \sum_{i=1}^{n} \frac{\alpha}{\beta}(1-e^{-\beta(T-t_i)})$$

Its partial deviates are the following:

$$\frac{\partial \mathcal{L}}{\partial \lambda_0} = \sum_{i=1}^{n} \frac{1}{\lambda_i} - T$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \sum_{i=1}^{n} \frac{1}{\lambda_i}\frac{\lambda_i-\lambda_0}{\alpha} - \sum_{i=1}^{n} \frac{1}{\beta}(1-e^{-\beta(T-t_i)})$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = -\sum_{i=1}^{n} \frac{\alpha}{\lambda_i} \sum_{j<i}(t_i-t_j)e^{-\beta(t_i-t_j)} - \frac{\alpha}{\beta}\sum_{i=1}^{n}\Big(-\frac{1}{\beta}(1-e^{-\beta(T-t_i)}) + (T-t_i)e^{-\beta(T-t_i)}\Big)$$

For an optimized calculus of the gradient, we note: $u_i = \sum_{j<i}(t_i-t_j)e^{-\beta(t_i-t_j)}$,and we compute $u_i$ by the induction relation:

$$\begin{aligned} v_0 &= u_0 = 0 \\ v_{i+1} &= e^{-\beta\delta_i}(v_i+1) \\ u_{i+1} &= u_i e^{-\beta\delta_i} + \delta_i v_{i+1} \end{aligned}$$

where $\delta_i = t_{i+1} - t_i$

We use the function `fmin_tnc` of module `scipy.optimize` that takes into argument a function and its gradient to compute the minimum.

For a simulated process with parameters $\lambda_0 = 1.2$, $\alpha = 0.6$, $\beta = 0.8$, we obtained the following results with mean on 100 values for $T = 100$, $T = 1000$ and with a single value for $T = 86400$:

| T | $\lambda_0$ | $\alpha$ | $\beta$ |
|---|---|---|---|
| 100 | 1.383 | 0.593 | 0.858 |
| 1000 | 1.243 | 0.598 | 0.806 |
| 86400* | 1.178 | 0.605 | 0.800 |
| Real values | 1.2 | 0.6 | 0.8 |

We conclude that we have a satisfying accuracy for our estimator. We can therefore use it for real data on tweets if we have a machine that can handle the huge size of the data: 5900 tweets per second! Nevertheless, if we had access to the data, we can change the time unit and use our estimator for a total time of 1 second with a regular computer.

## References

[1] Hawkes Processes, Patrick J. Laub, Thomas Taimre, Philip K. Pollett, July 13, 2015

[2] Y. Ogata. On lewis' simulation method for point processes. Information Theory, IEEE Transactions on, 27(1) :23–31, 1981.

[3] A.G.Hawkes. Spectraofsomeself-excitingandmutuallyexciting point processes. Biometrika, 58(1) :83–95, 1971.

[4] Queues Driven by Hawkes Processes, Andrew Daw, Jamol Pender, Cornell University, May 10, 2018

[5] A. Simma and M.I. Jordan. Modeling events with cascades of Poisson processes. arXiv preprint arXiv :1203.3516, 2012.