Automatic Spike Train Analysis and HTML Report Generation. An implementation with R, R2HTML and STAR.

Christophe Pouzat and Antoine Chaffiol

Laboratoire de Physiologie Cérébrale, CNRS UMR 8118
UFR biomédicale de l'université Paris-Descartes
45, rue des Saints-Pères
75006 Paris, France

November 9, 2007

Abstract

Multi-electrode arrays (MEA) allow experimentalists to record extracellularly from many neurons simultaneously for long durations. They therefore often require that the data analyst spends a considerable amount of time first sorting the spikes, then doing again and again the same basic analysis on the different spike trains isolated from the raw data. This spike train analysis also often generates a considerable amount of figures, mainly diagnostic plots, that need to be stored (and/or printed) and *organized* for efficient subsequent use. The analysis of our data recorded from the first olfactory relay of an insect, the cockroach *Periplaneta americana*, has led us to settle on such "routine" spike train analysis procedures: one applied to spontaneous activity recordings (typically epochs of 60s in the absence of any stimulation), the other used with recordings where a stimulation was repetitively applied (typically 15 to 20 epochs of 10 to 15 s with an odor puff). We have developed a group of functions implementing procedures commonly found in the literature and producing graphical or numerical outputs. These functions can be run in batch mode and do moreover produce an organized report of their results in an HTML file. A R package: STAR (Spike Train Analysis with R) makes these functions readily available to the neurophysiologists community. Like R, STAR is open source and free. We believe that our basic analysis procedures are of general interest but they can also be very easily modified to suit user specific needs.

Contents

1 Introduction

Multi-electrode arrays (MEA) allow experimentalists to record extracellularly from many neurons simultaneously for long durations. They therefore often require that the data analyst spends a considerable amount of time first sorting the spikes, then doing again and again the same basic analysis on the different spike trains isolated from the raw data. Although this "basic" analysis is likely to change from person to person, it will usually include, for "spontaneous activity data", a "laundry list" looking like:

- A display of the spike train per se in a *raster plot* or a *counting process plot* [11,28].
- A plot or a numeric quantity testing the stationarity of the train.

- Perhaps some standards distributions are fitted to the *inter spike intervals* (*isi*) and a plot checking the quality of the fits is produced.
- If several neurons are recorded simultaneously then *cross-correlation histograms* [24, 4] are likely to be generated.

Then depending on the results of this *systematic and preliminary analysis* the data analyst will decide to go further or to stop. In this scenario two issues arise:

- A lot of time ends up being spent doing fundamentally the same thing on different data. That is a strong incentive for automatization/batch processing.
- A lot of analysis results in the form of numerical summaries and graphics are being generated calling for a way to organize and display them in a systematic manner.

We insist here on the notion of *preliminary* analysis as opposed to the "refined" one which ends up being presented and illustrated in papers. There is still a long way between the preliminary and final analysis requiring a major input from the data analyst. The idea is to save time and stay alert for the really important part of the analysis.

In any case, even if the data analysis software used includes routines or functions to implement the individual components of our "laundry list", making the analysis automatic, *i.e.*, suitable for processing in batch mode, can be problematic. Difficulties arise from two sources:

- Getting good initial guesses for the optimization routines "doing the fits" can be tedious.
- Setting some "smoothing parameters", like a bin width, for plots is easily done by a human trying out several values and looking at the result but is a hard task for a "blind" computer.

We solved these two problems by:

- Using model reparametrization in the fitting routines [2] making the optimization step more robust with respect to "bad" specifications of initial guesses.
- Using "statistical smoothing" techniques based on regression splines [17, 31, 27] for the plots.

Once these two problems have received a satisfying answers the question of a suitable software environment into which these solutions will be implemented has to be answered. We chose R[1] [26] because, among many other reasons:

- It is open source and free.
- It runs on any computer likely to be found in a physiology laboratory, PC running Linux or Windows, Mac.
- It is a powerful and elegant programming language based on `Scheme` [14, 1].
- It uses state of the art numerical libraries (for optimization, random number generation, clustering, etc).
- It can be used in batch mode.
- It is, in our experience at least, incomparable for graphics[2].

[1] http://www.r-project.org

[2] See for instance: http://www.stat.auckland.ac.nz/~ihaka/787/ and http://addictedtor.free.fr/graphiques/index.php

- It is specifically designed to implement a clear and thorough type of data analysis [7].
- It is very easily extended by its users and, so to speak, encourages its users to become its programmers [8].

Adopting R did moreover provide a pretty straightforward solution to our "analysis results organization and display" problem. Modern computers are all equiped with a web browser and everyone knows how to use such a software. HTML files, the type of files that a web browser displays, can, as everyone knows, contain text, figures, mathematical equations and more. It seems therefore reasonnable to use the HTML format to organize our analysis results. If one agrees on that, the only problem left is the generation of this analysis specific HTML file. Well, the good news here is that the problem is solved by one of the R user contributed add-on packages: R2HTML[3] [18], which turns out to be very simple to use.

Equiped with these tools: R, statistical smoothing plus model reparametrization and R2HTML, we have developed a new R add-on package: STAR[4] (Spike Train Analysis with R) which like R itself is open source and free. The package contains 91 functions, most of which are seldomly *directly* used. It has now reached a satisfying maturity when applied to our data recorded from the antenal lobe (first olfactory relay) of the cockroach *Periplaneta americana*. We think that STAR could be useful to others and would like to know in any case how it works on different types of data. Because STAR is open source and because R makes it very easy for users to develop their own functions, we are confident that it could be adapted in a short time to other preparations.

2 Methods

2.1 Animal preparation and recordings

2.1.1 Animal preparation

Adult male cockroaches, *Periplaneta americana* were used as experimental animals. They were reared in an incubator with free access to food and water, at 25 °C. They were cold-anesthetized prior to the experiment. Wings, legs and some mouth parts were removed. Each insect was restrained in an acrylic glass holder, with its head fixed with dental wax. The lower part of both Antennae was protected with plastic tubes (to avoid contact with the physiological solution). A window of head cuticle was opened, the tracheae on the anterior face of the brain and the sheath surrounding the antennal lobes were removed. The esophagus was cut to reduce brain movement. Fresh cockroach saline was superfused on the brain. The saline composition was: NaCl 130 mM, KCl 12 mM, CaCl2 6 mM, MgCl2 3 mM, glucose 23 mM, HEPES 4mM; PH 7,2; 360 mosmol/kg.

2.1.2 *In vivo* recordings

MEA recordings were made in the antennal lobe using 16 sites silicon electrodes (Neuronexus Technologies). The probe was gently inserted into the antennal lobe until activity appeared at least on 4 recording sites. Signals were sampled at 12.8 kHz, band-pass filtered between 0.3 and 5 kHz and amplified using an IDAC2000 amplifier and its Autospike 2000 acquisition program (SYNTECH).

[3]http://www.stat.ucl.ac.be/ISpersonnel/lecoutre/R2HTML/

[4]http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR.html

2.1.3 Olfactory stimulations

A main moistened and filtered airflow was placed 3 cm away from the antenna, and a secondary stream controlled by a solenoid valve was used to deliver odor puffs. A piece of filter paper (3 mm x 20 mm) was soaked by 5 μl of pure aromatic compounds and placed in the secondary stream. 0.5 s odor puffs were used.

2.2 Data analysis

All the data analysis described in this paper was carried out using R [26], some of its user add-on packages and two additional packages developed by us, SpikeOMatic[5] and STAR[6]. "R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS."[7] The main subject of this paper is a desciption of some of the features of STAR. Sec. A.1 describes briefly how to obtain and install R and STAR.

2.3 Getting the spike trains: spike sorting

Spike sorting was carried out as described in: "The New SpikeOMatic Tutorial" [25].

2.4 Spontaneous activity analysis

We start with the analysis of "spontaneous regime" data. By that we mean data recorded in the absence of stimulation.

2.4.1 Spike train plot

The most common way to display a spike train is probably the *raster plot*, which is fundamentally a one dimensional graph where the occurrence time of the spike gives the horizontal coordinate and where a symbol like a little vertical bar, or a star ("*") is used to represent each spike. As abundantly illustrated by [28, Turnbull et al, 2005] and, in fact, proposed much earlier in the first figure of the book of [11, Cox and Lewis, 1966], it is much more informative to plot the cumulative number of events as a function of their occurrence time as shown on Fig. 1, where a classical raster plot is also added at the bottom of the graph. With this representation we can see the discharge dynamics much more clearly. When the firing rate increases it becomes difficult (if not impossible) to see the individual spike symbols on the raster and the capacity to distinguish between a moderate and a large increase in firing rate is strongly compromised. For instance on the raster of Fig. 1 the burst of spikes coming just after second 30 is barely distinguishable from the one coming at the end of the recording epoch, while on the cumulative plot we clearly see that the slope is much smaller for the second than for the first burst implying that the firing rate is smaller in the second than in the first burst. The increments of the cumulative plots during a burst give us, by definition, the number of spikes in the burst. The long burst coming after the 20th second is for instance made of roughly 100 spikes. We also see on the figure that their are no regular pattern of increase during a burst, implying that the successive bursts are made of a variable number of spikes. Clearly, we can say a lot more about a spike train by looking at a cumulative plot rather than at a raster plot. In addition important non-stationarities of the discharge will show up as a

[5]http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/newSOM/newSOMtutorial/newSOMtutorial.html.

[6]http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR.html.

[7]Quotation from the home page of the "The R Project for Statistical Computing": http://www.r-project.org/

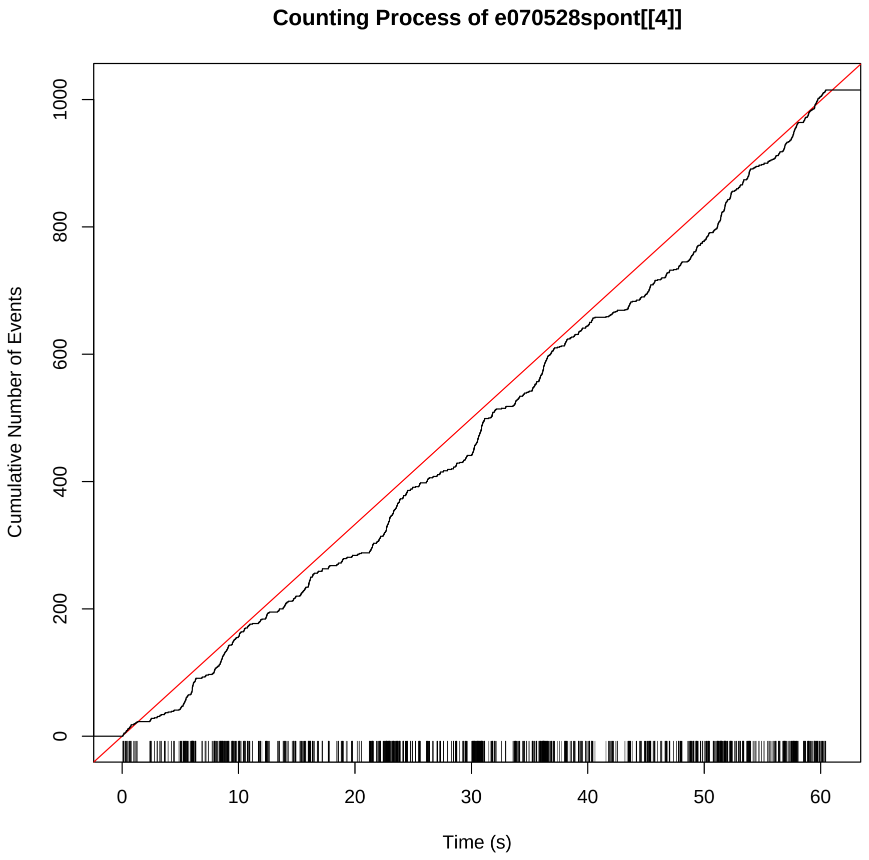


Figure 1: A spike train plot of neuron 4, data set, e070528, spontaneous regime. The generation of this figure with STAR is explained in Sec. A.8.

curvature of the graph, which will be concave for a decelerating discharge and convex for an accelerating one.

We refer to the kind of plot shown on Fig. 1 as a *counting process plot* because that what such plots are called in the (statistical) literature dealing with series of (identical) events [3, 15]. A *Counting process* is a right continuous step function which undergoes a unit jump every time an event occurs. More formally [3, Brillinger, 1988, p190, Eq. 2.1]: For points $\{t_j\}$ randomly scattered along a line, the counting process $N(t)$ gives the number of points observed in the interval $(0, t]$.

$$N(t) = \sharp\{t_j \text{ with } 0 < t_j \leq t\} \tag{1}$$

where $\sharp$ stands for the cardinality (number of elements) of a set and where the $\{t_j\}$ stand the the occurrence times of the spikes. As we will later see, our discharge models will in fact end up predicting $N(t)$ (or at least try to).

2.4.2 Poisson process

We have just mentioned that we want to model spike discharges. In the simplest cases these discharges would be well aproximated by a *homogenous Poisson process*, which is formally defined as a stochastic process fully characterized by a *time independent rate parameter*, λ, such that the number of events observed in $(t, t+\tau]$ follows a *Poisson distribution* with parameter $\lambda\tau$ (for all $t > 0$). Using our previous *Counting process* definition (Eq. 1) we would write:

$$\text{Prob}\{N(t+\tau) - N(t) = n\} = \frac{(\lambda\tau)^n}{n!} \exp(-\lambda\tau) \tag{2}$$

The additional requirement of independence of the observed counts, n_1 and n_2, on two *non-overlapping* time intervals: $(t_1, t_1 + \tau_1]$ and $(t_2, t_2 + \tau_2]$, defines a *homogenous Poisson process*.

The *homogenous Poisson process* is not very useful to analyze "directly" real spike trains, at least not ours, but following [5, Brown et al., 2002] and even more [21, Ogata, 1988], it is extremly useful to judge the adequacy between fitted discharge models and the data at hand. To make such a use of it we are going to need few additional properties of the *homogenous Poisson process* that we state next for completeness.

It can be shown [22, Pelat, 1996, Chap. 9] that the above two conditions (Poisson distribution of the counts and independence) are *equivalent* to the following two requirements:

- The process is *orderly* , that is:

$$\lim_{\tau \to 0} \frac{\text{Prob}\{N(t+\tau) - N(t) > 1\}}{\text{Prob}\{N(t+\tau) - N(t) = 1\}} = 0 \tag{3}$$

  In words, orderliness implies that events occur separately at most one at a time (as opposed to clustered events). Given the refractory period exhibited by neuronal discharges we should not have to worry to much about fulfilling this assumption in practice (unless we have a serious spike sorting problem).

- The distribution of time intervals, I, between successive events is *exponential with constant* $\frac{1}{\lambda}$, *i.e.*, its *probability density function* is:

$$\text{p}(i) = \lambda \exp(-\frac{i}{\lambda}) \tag{4}$$

The last property we are going to use is [11, Cox and Lewis, 1966, Chap. 2]: If a *homogenous Poisson process* with rate $\lambda = 1$ is observed during a time T and gives rise to K events, then the cumulative distribution function of the event times, $\{t_i\}_{i=1}^{K}$, is linear with slope $\frac{1}{T}$ on $(0, T]$, is zero below 0 and 1 above T.

2.4.3 Renewal test plot

When a *homogenous Poisson process* is not good enough for the data, the next type of models to try is the *homogenous renewal process*. A homogenous renewal process is a process where the intervals between successive events, the *inter spike interval*s, come all *independently* from the *same* distribution (they are said to be *independent and identically distributed* or *iid*). In other words it is enough to characterize the *isi* distribution (together with the distribution of the first event) to fully characterize the whole process.

The definition of a *homogenous renewal process* leads us quickly to a graphical test that such processes should pass. Let as before, $\{t_j\}_{j=1}^{K}$, be our K spike times from which we get $K-1$ *isi*s: $\{isi_j = t_{j+1} - t_j\}_{j=1}^{K-1}$. We can sort these *isi*s in increasing order to get: $\{i_{(j)}\}$, where $i_{(j)} \leq i_{(l)}$ if $j < l$. Let O_j be the rank of interval i_j of the orginal (unsorted sequence) in the new sorted one. To use a concrete example, let us asume that we have the following original sequence of 5 *isi*s (expressed in s):

```
    1     2     3     4     5
0.031 0.062 0.073 0.092 0.054
```

Then the $\{O_j\}$ and the sorted, $\{i_{(j)}\}$, sequences are:

```
    1     5     2     3     4 
0.031 0.054 0.062 0.073 0.092
```

Now if the $\{i_j\}$ are *iid* then the $\{O_j\}$ should be very nearly so (in the sense that the joint distribution of (O_j, O_{j+k}) should be uniform on $\{1, \ldots, K-1\} \times \{1, \ldots, O_j - 1, O_j + 1, \ldots, K-1\}$ for $k \neq 0$). Then if we plot O_{j+1} as a function of O_j we should see the square $\{1, \ldots, K-1\} \times \{1, \ldots, K-1\}$ uniformly filled, without pattern.

By plotting O_{j+1} as a function of O_j instead of i_{j+1} as a function of i_j we are making a better use of the surface of the plot and that is not an aesthetic issue but a way to make the plot more informative. We can then subdivide the surface defined by the $\{1, \ldots, K-1\} \times \{1, \ldots, K-1\}$ square into subsquares and apply a χ^2 test to the contingency table so defined. This is what we systematically do with our data. We generate two plots: $O_{j+1}\,\mathrm{vs}\,O_j$ and $O_{j+2}\,\mathrm{vs}\,O_j$. We subdivide the surface of the plots into squares of identical surface. The surface is chosen per default[8] so that under the null hypothesis of independence, at least 25 events would fall into into each square (this is to insure the applicability of the χ^2 test). We do this χ^2 statistics computation not only at lag 1 and 2 but also up to lag: $10 \log_{10}(K-1)$[9] (this maximal lag can be set by the user). We then also plot this χ^2 statistics as a function of the lag and show the 95% confidence region of the χ^2 in the background.

We also plot the *isi* empirical autocorrelation function, what [23, Perkel et al, 1967] call the *serial correlation coefficients* function[10], and test it against the null hypothesis of no correlation.

Fig. 2 shows what we call a "renewal test plot" from which it is rather clear that the *homogenous renewal process* model does not apply. These renewal test plots are, in our experience, also very sensitive to non-stationarities which makes sense given that a *homogenous renewal process* must be stationary by definition. The reader is invited to explore this with one of the demonstrations of STAR. Sec. A.10 describes how to do that.

2.4.4 Bivariate duration distributions fits

In addition to the renewal test plot we have included in our automatic spike train analysis procedures a systematic fit of the empirical *isi* distribution with 6 common bivariate duration distributions [19] whose *probability density functions* (*pdf*s) are:

- log normal:
$$\mathrm{dlnorm}(i; \mu, \sigma^2) = \frac{1}{\sqrt{i 2\pi\sigma^2}} \exp\Big(-\frac{1}{2}\frac{(\log i - \mu)^2}{\sigma^2}\Big) \tag{5}$$
- inverse Gaussian:
$$\mathrm{dinvgauss}(i; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2 i^3}} \exp\Big(-\frac{1}{2}\frac{(i - \mu)^2}{i\sigma^2\mu^2}\Big) \tag{6}$$
- gamma:
$$\mathrm{dgamma}(i; \alpha, \beta) = \frac{i^{\alpha-1}}{\beta^\alpha \Gamma(\alpha)} \exp(-\frac{i}{\alpha}) \tag{7}$$
- Weibull:
$$\mathrm{dweibull}(i; \alpha, \beta) = \frac{\alpha}{\beta}(\frac{i}{\beta})^{\alpha-1} \exp(-(\frac{i}{\beta})^\alpha) \tag{8}$$

[8]See the documentation of the renewalTestPlot function of STAR

[9]$\log_{10}$ stands for the decimal logarithm, *i.e.*, $\log_{10}(10) = 1$.

[10]Defined in their Eq. 7 and 8, p 400

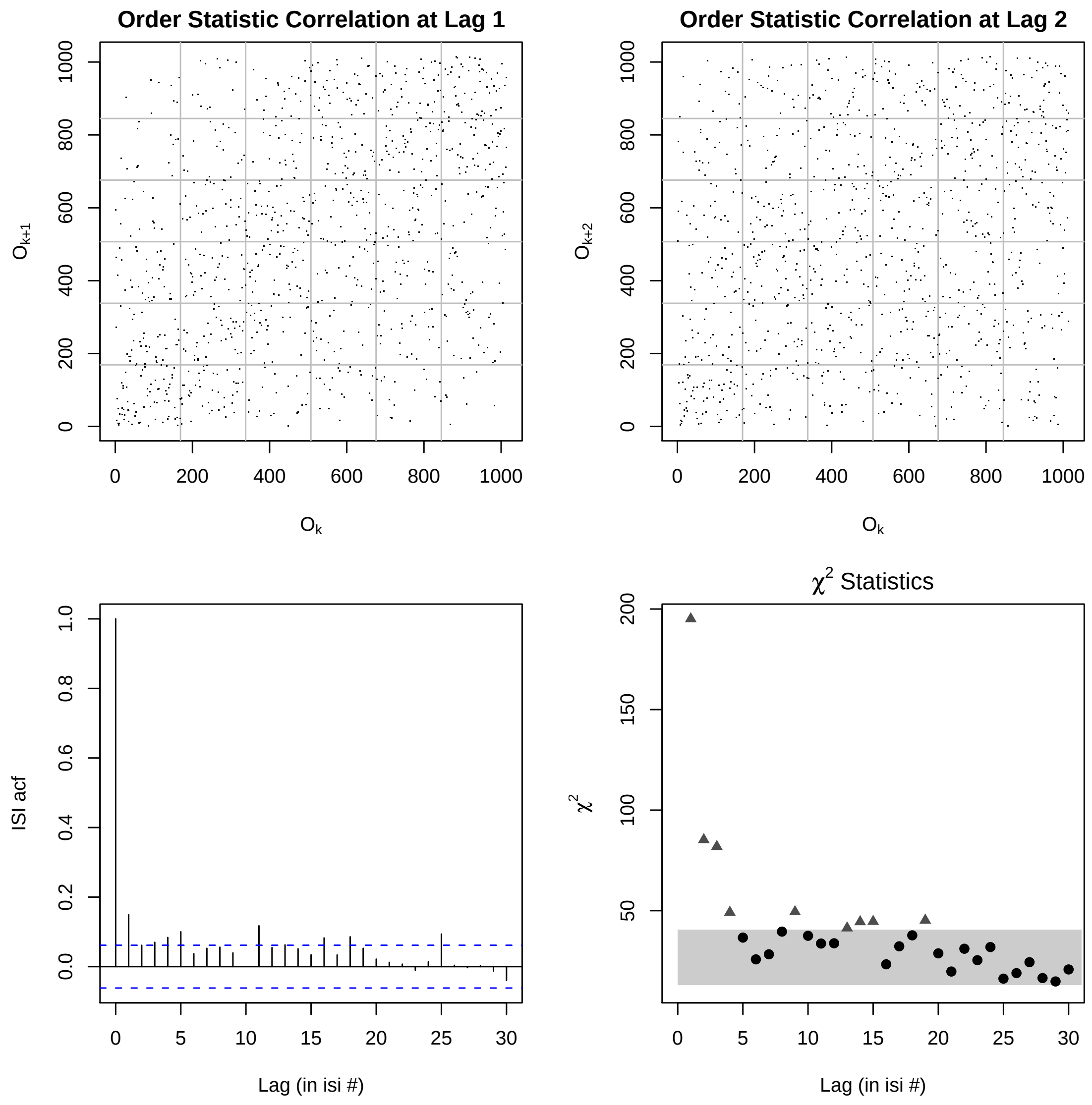


Figure 2: A “renewalTestPlot” of neuron 4, data set, e070528, spontaneous regime. The generation of this figure with STAR is explained in Sec. A.10.

- refractory exponential:

$$\mathrm{drexp}(i;\alpha,i_{min}) = \begin{cases} 0 & \text{if } i < i_{min} \\ \alpha\ \exp(-\alpha\,(i-i_{min})) & \text{if } i \geq i_{min} \end{cases} \tag{9}$$

- log logistic:

$$\mathrm{dllogis}(i;\mu,\sigma) = \frac{1}{i\sigma}\frac{\exp(-\frac{\log i-\mu}{\sigma})}{\left(1+\exp(-\frac{\log i-\mu}{\sigma})\right)^2} \tag{10}$$

The names we have used for these *pdf*s like, dlnorm, are the names under which these functions can be called in R. Parameters estimates for these distributions are obtained by the *maximum likelihood* method [16]. The ones of the lognormal, inverse Gaussian and refractory exponential distributions are available in closed form [19]. The ones of the gamma, Weibull and log logistic distributions are obtained by numerical optimization using function optim for the Weibull and the log logistic distributions and using the profile likelihood method for the gamma distribution [20, Monahan, 2001, pp 210-216]. Initial guesses are obtained by the method of moments. In addition reparametrization is used systematically for parameters which are contrained to be positive [2,20], like the two parameters of the gamma distribution. For those the *log likelihood function* is written in term of the log parameters. See the examples of gammaMLE, weibullMLE and llogisMLE for details.

Once some or all of these distributions have been automatically fitted the capicities of the models to fit the data are compared with the *Akaike's Information Criterion* [6,19]. The *Akaike's Information Criterion* selects the best model among a set of models but does not provide any clue regarding how well the best model fits the data. A way to do that is build a *theoretical quantile quantile plots* [9, Chambers et al, 1983, Chap. 6] as shown on Fig. A.11. On these plots, a perfect fit would fall on the diagonal which is drawn in red. 95 and 99% pointwise confidence intervals are also drawn[11].

2.4.5 A more general goodness of fit test

We have just used *theoretical quantile quantile plots* to test the adequacy of our fitted duration distribution models to our data. The problem of these tests is that they are not easily generalized when one starts entertaining more complicated discharge models, like models involving the length of the previous *isi*, or of several previous *isi*s, models including a stimulus effect, models including functional coupling between neurons. To this end but in a different context, [21, Ogata, 1988] introduced a battery of goodness of fit tests that a good "discharge model"[12] should pass. One of these tests was brought into neurophysiology by [5, Brown et al, 2002]. The tests are all based on a "time transformation" (Ogata's terminology) or "time rescaling" (Brown et al). The idea of time transformation for spike train models had in fact already been used by [15, Johnson, 1996] who used it for simulation purposes.

We are going to illustrate the time transformation idea and the tests it makes available assuming that one of our 6 duration distribution model is the good one (like the inverse Gauss model for the neuron illustrated in Fig. 3). We will write $f(i)$ the *pdf* of the *isi*s of our neuron. To this *pdf* we can associate [23, Perkel et al, 1967, Eq. 4-6, pp 396-397]:

- The *cumulative distribution function*:

$$\mathrm{Prob}\{T \leq t\} = F(t) = \int_0^t f(i)di \tag{11}$$

[11]To be honest we have to say that Fig. A.11 is one of our rare examples where one of 6 duration distribution (the inverse Gaussian) is able to fit our data.

[12]Ogata worked on earthquakes sequences, so his events were earthquakes, not spikes.

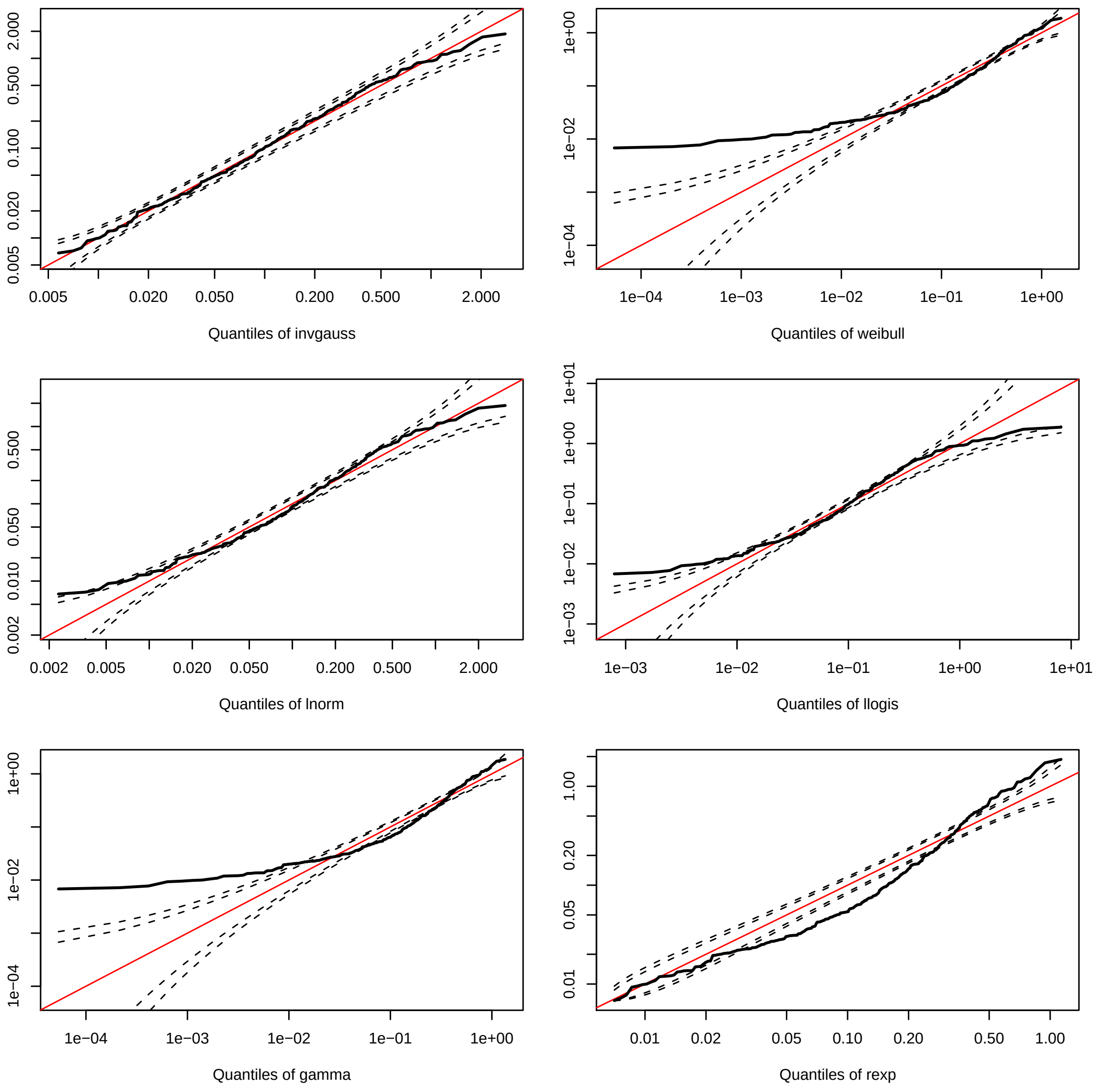


Figure 3: *theoretical quantile quantile plot*s of the six fitted duration distribution to the *isi*s of neuron 1, data set, e070528, spontaneous regime. The generation of this figure with STAR is explained in Sec. A.11.

- The *survivor function*:

$$\mathrm{Prob}\{T > t\} = S(t) = 1 - F(t) \tag{12}$$

- The *intensity function*:

$$\lambda(t) = \frac{f(t)}{S(t)} \tag{13}$$

  The *intensity function* gives the probability of an event at time t *given that no event occurred between 0 and t*. It can be interpreted as the intantaneous rate of events. Moreover, $\lambda > 0$ on the *support* of f, that we shall note: $\mathbb{R}_f$ and define as:

$$\mathbb{R}_f = \{t \in \mathbb{R}^+ \,\text{such that}\, f(t) > 0\} \tag{14}$$

  The *intensity function*, is also called: *the hazard function*, the *the conditional intensity function* [3], the *stochastic intensity function*.

Using Eq. 11 and 12 one rewrites Eq. 13 as:

$$\lambda(t) = \frac{-\frac{dS(t)}{dt}}{S(t)}$$

which, using the fact that: $S(0) = 0$ (from Eq. 12), leads to:

$$S(t) = \exp\big(-\int_0^t \lambda(\tau)d\tau\big) \tag{15}$$

We are going to use the integral of the *intensity function* a lot so it is worth introducing a new function, the *integrated intensity function* for it:

$$\Lambda(t) = \int_0^t \lambda(\tau)d\tau \tag{16}$$

The properties of $\lambda(t)$ imply that $\Lambda(t)$ is a one-to-one function: $t \in \mathbb{R}_f \mapsto \Lambda(t) \in \mathbb{R}^+$. Using Eq. 16 we rewrite Eq. 11 as:

$$F(t) = 1 - \exp(-\Lambda(t)) \tag{17}$$

We then see that if we transform / rescale the time with Λ, the *cumulative distribution function* of a transformed time ($t \mapsto \Lambda(t)$) is the *cumulative distribution function* of a *homogenous Poisson process* with a rate 1.

Let us now assume that we have observed the sequence of spike times, $\{t_j\}_{j=1}^K$ between 0 and T. To simplify the formalism we are going to do as if we had observed from t_1 to t_K[13]. Using λ, we map recursively $\{t_j\}_{j=1}^K$ onto $\{\Lambda_j\}_{j=1}^K$ as follows:

$$\begin{aligned}\Lambda_1 &= 0 &(18)\\ \Lambda_j &= \Lambda_{j-1} + \int_0^{t_j - t_{j-1}} \lambda(t)dt\,, \quad \forall\, j \in \{2,\ldots,K\} &(19)\end{aligned}$$

where λ is defined by Eq. 13.

But from Eq. 17 the intervals between our transformed times $\{\Lambda_j\}_{j=1}^K$ have an exponential distribution and by construction we cannot have two or more events at the same transformed time (because we started with strictly increasing t_js, otherwise we have to worry about our spike sorting quality!), therefore from the result stated in Sec.2.4.2, *the*

[13] This does induce a very small bias of the estimates but it can be typically ignored when we have more that, say, 50 spikes. Which is always the case for our data.

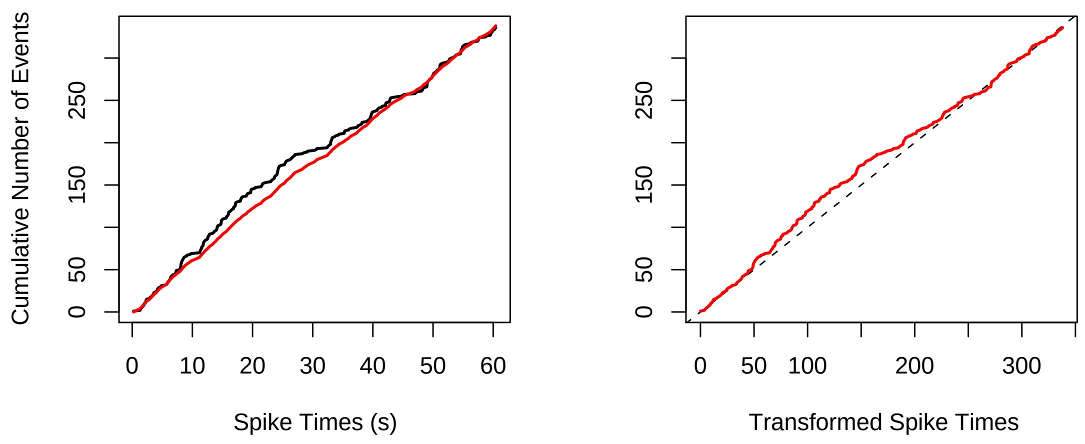


Figure 4: Illustration of the time transformation / rescaling. Neuron 1 of data set e070528 (spontaneous regime) was fitted with an inverse Gauss distribution and time transformation was performed following Eq. 18 and 19. The left hand plot shows the *Counting process* (black) and the transformed time (red) as a function of actual spike times. The right hand plot shows the *Counting process* as a function of transformed time (red). The diagonal appears as a black dotted line. Sec. A.14 explains how to generate this plot.

transformed sequence, $\{\Lambda_j\}_{j=1}^K$, *is the realization of a homogenous Poisson process with unit rate*. This is why the *homogenous Poisson process* is indirectly so important also for the analysis of real spike trains.

Fig. 4 illustrates the time transformation / rescaling. The left hand plot shows the *Counting process* and the $\{\Lambda_j\}_{j=1}^K$ as functions of spike times, the right hand side shows the *Counting process* as a function of the $\{\Lambda_j\}_{j=1}^K$. The key feature here is that when the model is good, the $\{\Lambda_j\}_{j=1}^K$ *predicts* the *Counting process*. The $\{\Lambda_j\}_{j=1}^K$ being, when the model is correct, the realization of a *homogenous Poisson process* with rate 1, its value at any transformed time is the expected cumulative number of events. The variance of events counts should moreover be equal to the expected value.

Pushing farther the implications of the theoretical *homogenous Poisson process* with unit rate after time transformation, we get the four major tests of Ogata [21]:

- The result stated at the end of Sec. 2.4.2 imply that the $\{\Lambda_j\}_{j=1}^K$ should ideally fall on the diagonal of the right hand plot of Fig. 4. The deviations form diagonal likely to happen can be quantified with a Kolmogorov-Smirnov test at 95 and 99%.

- the u_k defined, for $k > 1$, by:

$$u_k = 1 - \exp\big(-(\Lambda_k - \Lambda_{k-1})\big) \tag{20}$$

  should be *iid* with a uniform distribution on $(0, 1)$. A pattern in a plot of u_{k+1} vs u_k would be inconsistent with the *homogenous Poisson process* hypothesis.

- The empircal *cumulative distribution function* of the sorted $\{u_k\}$ can be compared to the *cumulative distribution function* of the null hpothesis (*iid* with a uniform distribution on $(0, 1)$) with a Kolmogorov-Smirnov test. This test is attributed to Berman by Ogata [21] and is the one proposed and used by [5, Brown et al, 2002].

- We can split the transformed time axis into K_w non-overlapping windows of the same size w, count the number of events in each window and get a mean count N_w and a variance V_w computed over the K_w windows. Using a set of increasing window sizes: $\{w_1, \ldots, w_L\}$ we can plot V_w as a function of N_w and check if the result falls on a straight line going through the origin with a unit slope, as expected if the $\{\Lambda_j\}_{j=1}^{K}$ are the realization of a *homogenous Poisson process* with rate 1.

Fig. 5 shows the "Ogata's tests battery" in action using the case already illustrated by Fig. 3 and 4.

2.4.6 Cross correlation histograms

The *cross-correlation histograms* functionalities we have included in STAR are very close to what is described under this name in [4, Brillinger et al, 1976]. The main difference is that we don't use the square root transformation to stabilize the variance since there is no variance to stabilize under the null hypothesis of no correlation.

In addition STAR provides a *smooth cross-correlation histogram* which is build exactly in the same way as the *smooth peri stimulus time histograms* of Sec. 2.5.2, where they are described in detail. Fig. 8 shows a screen shot of an automatically generated report where both types of *cross-correlation histograms* applied to *the same* data can be seen.

2.5 Stimulus response analysis

The next considered case is the analysis of repeated stimulations like, in our case, 15 to 20 successive applications of an odor puff to the antenna for 0.5 s every minute. We assume here that the data from a single neuron have been formated such that they are all locked with repesct to the stimulus onset. Considering that the actual number of spikes generated by a neuron during a fixed acquisition epoch using always the same stimulus will fluctuate, a matrix is not the proper format to store the spike trains of a neuron. The list object of R is, on the other hand, particularly well adapted for this task (see Sec. A.6 for details about R list objects).

2.5.1 Raster plot

Although *counting process plots* can be generalized to display successive responses of a neuron to repeated stimulations we found the more conventional *raster plot* better suited for this task. This is therefore the default method we use to display "raw" stimulus response data. In our automatic spike train analysis and report generation procedure we moreover add to it a smooth version of the classical *peri stimulus time histogram* [13]. We discuss next these *smooth peri stimulus time histograms*.

2.5.2 Smooth PSTH

The case for using smooth as opposed to "classical" *peri stimulus time histograms* (*psth*) has been clearly and convincingly made by [17, Kass, Ventura and Cai, 2003] and won't be repeated here. We are instead going to illustrate the principle while underlying the difference in methodology. A *smooth peri stimulus time histogram* (*spsth*) is not only statistically more "efficient" than a *peri stimulus time histogram* but it also facilitates the development of an automatic spike train analysis software being not (or at least much less) sensitive to a bin width choice.

The *psth* (like the *spsth*) can be given a firm statistical basis as soon as one considers that the relevant feature of a single neuron response to a given stimulus is its "average" or "mean" response, that is, it's *average intantaneous firing rate*. For then if the successive

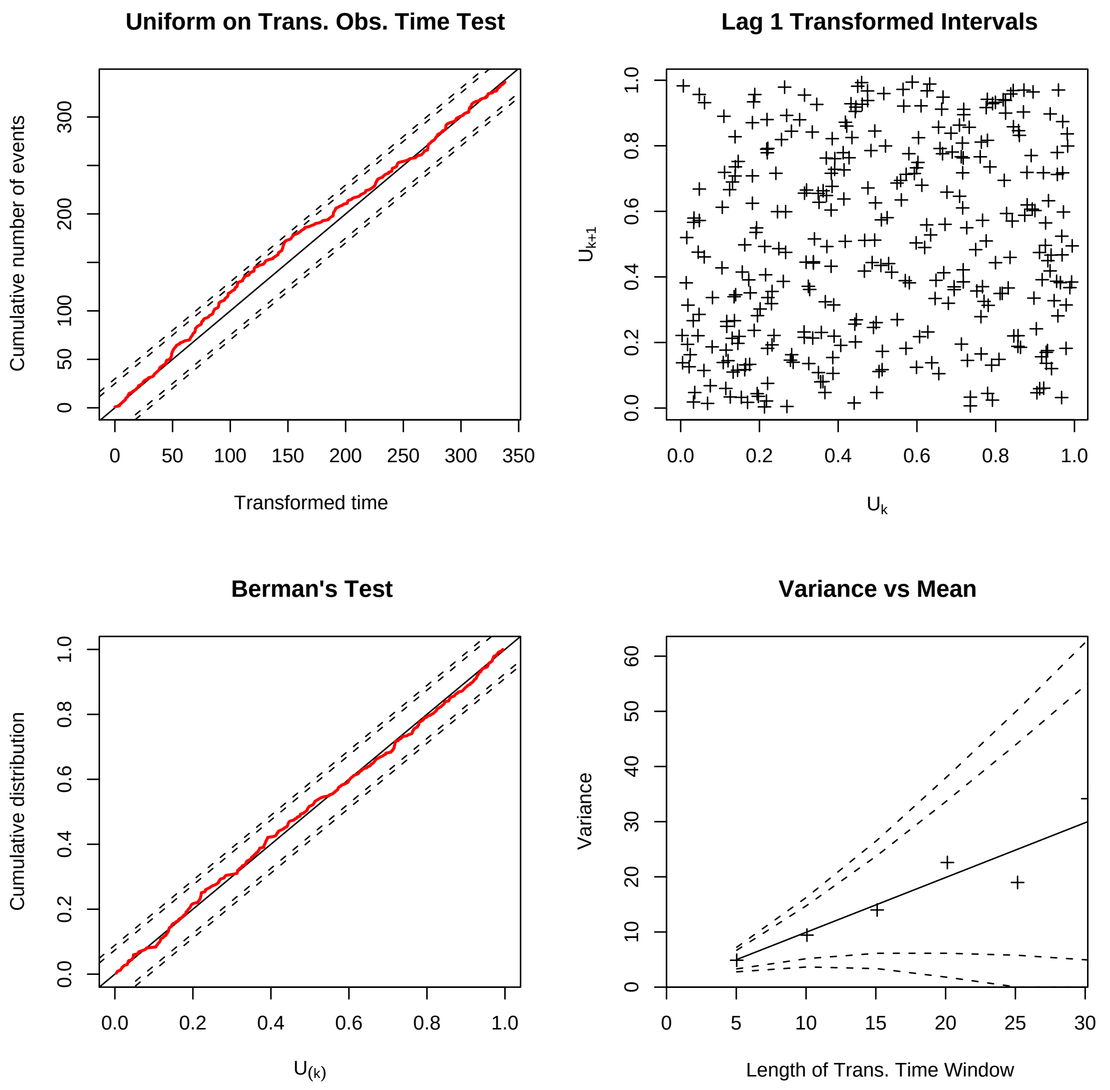


Figure 5: The Ogata's tests battery applied to neuron 1 of data set e070528 (spontaneous regime). The dotted lines on the left hand side plots define *confidence bands*, *i.e.*, 5 cases out of 100 should on average cross the inner boundaries and 1 out of 100 should cross the outer boundaries under the null hypothesis. The dotted lines on the "Variance vs Mean" plot (bottom right) define confidence intervals, *i.e.*, 5 points out of 100 (respectively 1 out of 100) should cross the inner (resp. outer) boundaries on average. Sec. A.15 explains how to generate this plot.

reponses of the neuron are uncorrelated and are "collapsed" or aggregated[14], the resulting process tends to an *inhomogenous Poisson process* [29, Ventura et al, 2002, Sec. 4, p6]. A *inhomogenous Poisson process* is like a *homogenous Poisson process* (Sec. 2.4.2) for which the rate λ is allowed to be time dependent. That is, Eq. 2 has to be modified as follows:

$$\text{Prob}\{N(t+\tau) - N(t) = n\} = \frac{\left(\int_t^{t+\tau} \lambda(u)\,du\right)^n}{n!} \exp\left(-\int_t^{t+\tau} \lambda(u)\,du\right) \tag{21}$$

where we see that if $\lambda(u) = \text{Cst}$, Eq. 2 is obtained. The traditional *psth* produces a λ estimate that is a step function (λ is supposed to be constant over the time period spanned by a bin). The *spsth* produces a smooth estimate of λ. In order to produce any of the two estimates, the "raw" data (the 15 to 20 spike trains of the considered neuron for a given stimulus) have to be pre-processed by binning. Then for a *psth*, the bin counts are divided by the number of trials and by the bin length to obtain the final estimate. Approximate confidence intervals are also available using the Poisson assumption. Ignoring the statistical efficiency issue, the crucial parameter is the bin width (or more generally the sequence of bin widths). If the width is set too large, sudden changes of λ will be missed (filtered out), if set too small statistical fluctuations (due to the finite sample size) might be confounded with genuine changes of λ. The traditional solution to this problem is *interactive*: try out several bin widths and see the results. Having confidence intervals clearly helps in making "objective" choices with such an approach. Fig. 14 illustrates this point.

A *smooth peri stimulus time histogram* produces an estimate, $\hat{\lambda}(t)$, of $\lambda(t)$ through a compromise between two antogonistic goals. If $\{x_i\}_{i=1}^N$ is the set of bin centers used in the preprocessing stage and if $\{y_i\}_{i=1}^N$ is the set of corresponding counts, the *log likelihood* of $\hat{\lambda}$ is:

$$\mathcal{L}(\hat{\lambda}) = \sum_{i=1}^{N} \Big(y_i \log\big(\hat{\lambda}(x_i)\big) - \hat{\lambda}(x_i)\Big) \tag{22}$$

Remember that the likelihood is *proportional* to the probability of the data. This *log likelihood function* is maximized by a step function whose step height in each bin is given by $\frac{y_i}{\delta}$. That is what the classical *psth* does. But we want here a *smooth* estimate of λ. To this end we are going to express $\hat{\lambda}(t)$ as a *linear combination* of a set of *given* basis functions:

$$\hat{\lambda}(t) = \sum_{j=1}^{q} b_j(t)\,\beta_j \tag{23}$$

where the b_js are smooth functions and the β_js are estimated parameters. A common, but not necessary, choice of b_js is a set of cubic polynomial in between pre-defined "knots" with a contraint enforcing continuity of the the function defined by Eq. 23 up to the second derivative: giving so called *cubic splines*. The approach known as *penalized regression splines* consists then in a choice of a large q (Eq. 23) combined with a penalty proportional to:

$$\int [\hat{\lambda}''(t)]^2 dt \tag{24}$$

Such a penalty avoids "too wiggly" $\hat{\lambda}$s. What is then *minimized* is:

$$-\sum_{i=1}^{N} \Big(y_i \log\big(\hat{\lambda}(x_i)\big) - \hat{\lambda}(x_i)\Big) + \rho \int [\hat{\lambda}''(t)]^2 dt \tag{25}$$

The problems to be solved then are:

[14]That is, if the trial of origin of the spikes is lost (or ignored), as happens upon building a *psth*.

- What is a good choise of basis functions: b_j, in Eq. 23.
- What is a good choice of q, in Eq. 23.
- What is a good choice of ρ, in Eq. 25

These problem are discussed remarkably clearly in Simon Wood's book [31] and are already well introduced in one of his papers [30]. Several approaches are possible, in addition to Simon Wood's approach implemented in his package mgcv, which is the one used in STAR, Kass and collaborators developed a Bayesian approach [12, 17]. Sec. A.18 gives a detailed account on how *spsth*s are built in STAR using mgcv.

Regardless of the method used, we notice that once $\hat{\lambda}$ is obtained it can be "fed to" Eq. 16 and used for a time transformation. After that the Ogata's tests battery can be applied as illustrated in Fig. 11. In that case the individual trials are time-transformed before being added together.

2.6 HTML report generation

Luckily the HTML report generation is the easiest part of this "Methods" section. HTML files are plain ASCII files where *tags* are used to control the way text, images, etc, appear on the browser. These tags are surrounded by "<>" symbols. This means that as soon one is using an analysis software which can append text to an ASCII file it becomes possible to write an HTML file from the analysis software. Of course R can do that and even better, we need to know very little about HTML in order to generate our reports in that format. Eric Lecoutre's package R2HTML [18] does everything (almost) for us. His short 2003 paper [18] is enough to get one going within 15 minutes.

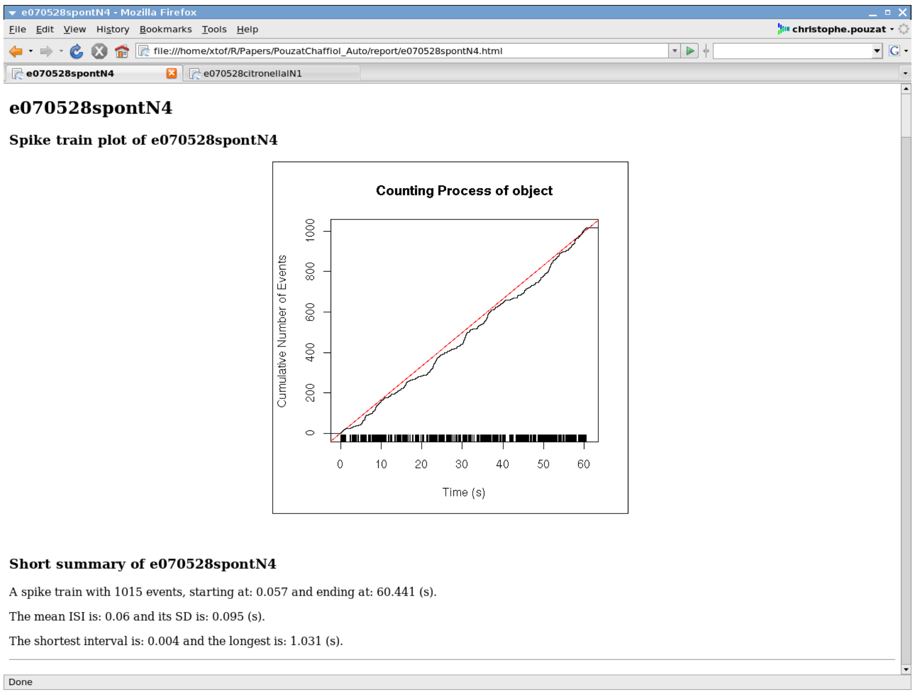
The "strategy" we followed to develop our reportHTML methods was first to write an R script, that is, a succession of R commands that was satisfying for what we wanted to do (basic analysis of spike trains). We then turned this script into a function so that arguments could be passed in order to adapt to, say, different data names. After that we used R2HTML function HTMLInitFile at the beginning of the function to open the resulting file, we used function HTML to write the specific computation results we wanted to include in the HTML file, while with function HTMLInsertGraph we were able to include graphs in the report. We ended up closing the HTML file with function HTMLEndFile. It could hardly be simpler.

3 Results

We illustrate here our two automatic spike train analysis procedures using 2 data sets from a single experiment. Four neurons were simultaneously recorded in this experiment. The "spontaneous regime" data set (e070528spont) is an epoch of 60 s of continuous acquisition (see ?e070528spont for details). We are going to illustrate the results obtained with neuron 4. The "stimulus response regime" data set (e070528citronellal) comes from 15 stimulations with citronellal (0.5 s puff) repeated every minute. Each acquisition epoch was 13 s long (see ?e070528citronellal for details). We are going to illustrate the results obtained with neuron 1. The reader is invited and encouraged to repeat the analysis results presented in this section and to do do the same analysis with the other neurons of the same data set as explained in the Appendix. These data sets, as well as three other ones, are part of the STAR package (see Sec. A.5 for details).

3.1 Spontaneous activity analysis

Our automatic spontaneous spike train analysis and report generation procedure, reportHTML.spikeTrain, performs sequentially the "sub-tasks" of Sec. 2.4. To avoid having


e070528spontN4 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
christophe.pouzat
file:///home/xtof/R/Papers/PouzatChaffiol_Auto/report/e070528spontN4.html
e070528spontN4
e070528citronellaN1
Counting Process of object
Cumulative Number of Events
0 200 400 600 800 1000
0 10 20 30 40 50 60
Time (s)
Done

e070528spontN4

Spike train plot of e070528spontN4

Short summary of e070528spontN4

A spike train with 1015 events, starting at: 0.057 and ending at: 60.441 (s).

The mean ISI is: 0.06 and its SD is: 0.095 (s).

The shortest interval is: 0.004 and the longest is: 1.031 (s).

Figure 6: Fig. 1 as it appears in the automatically generated HTML report.

too many figures of screen shots in this paper we do not show the whole report here but it can be seen on our web site[15] and it can be reproduce by the reader on his/her computer.

Briefly, a spike train plot (Sec. 2.4.1) is made and added to the HTML report. For instance, Fig. 1 can be seen again on the screen shot of the HTML report shown in Fig. 6.

Summary information including the number of spikes, the times of the first and last spikes, the mean *isi*, etc, are computed and added to the report as can be seen at the bottom of Fig. 6.

The renewal test plots (Sec. 2.4.3, Fig. 2) are built and added to the report (not shown on screen shots).

The six duration distributions are fitted (Sec. 2.4.4) and the best one is used to apply a time transformation to the spike train (Sec. 2.4.5).

The Ogata's tests battery is applied and if it passed at the the 99% confidence level (see ?summary.transformedTrain for details), the result of the transformation is plotted as well as all the *theoretical quantile quantile plots* of Sec. 2.4.4. If argument forceTT is set to TRUE (default), then these last two plots are added even if the best model does not pass the tests. Fig. 7 shows a screen shot of the report where the Ogata's tests battery appears. In contrast to the case illustrated in Fig. 5 it clear that the tests are not passed. The illustrated neuron is not (at all) well described by a *homogenous renewal process* model.

If other spike trains (from simultaneously recorded neurons) are provided, then *cross-correlation histograms* are estimated. Two estimations methods are available (Sec. 2.4.6), the classical histogram and a smooth version of it. Argument chh controls if a single estimation is performed or if both are performed. If the smooth version is requested a numerical summary of the fit is also printed in the report. Moreover if argument doGamCheck is set to TRUE then check plots (Sec. A.18 and Fig. 16) are added to the report. Fig. 8 shows a screen shot where the *cross-correlation histograms* built with neuron 4 as a reference and neuron 2 as a test. Confidence intervals at 95% level are shown on these two *cross-correlation histograms*. The excess of spikes of neuron 2 roughy 100 ms before

[15] http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR_folder/e070528spontN1.html

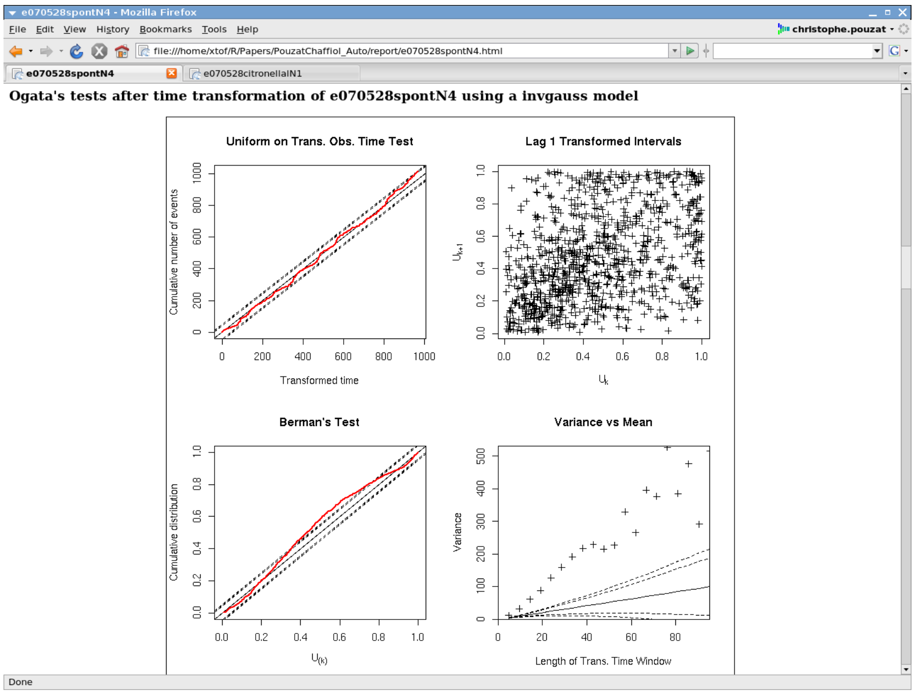

e070528spontN4 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
christophe.pouzat
file:///home/xtof/R/Papers/PouzatChaffiol_Auto/report/e070528spontN4.html
e070528spontN4
e070528citronellalN1
Ogata's tests after time transformation of e070528spontN4 using a invgauss model
Uniform on Trans. Obs. Time Test
Cumulative number of events
Transformed time
Lag 1 Transformed Intervals
U_{k+1}
U_k
Berman's Test
Cumulative distribution
$U_{(k)}$
Variance vs Mean
Variance
Length of Trans. Time Window
Done

Figure 7: The equivalent of Fig. 5 this time with neuron 4 as it appears in the automatically generated HTML report.

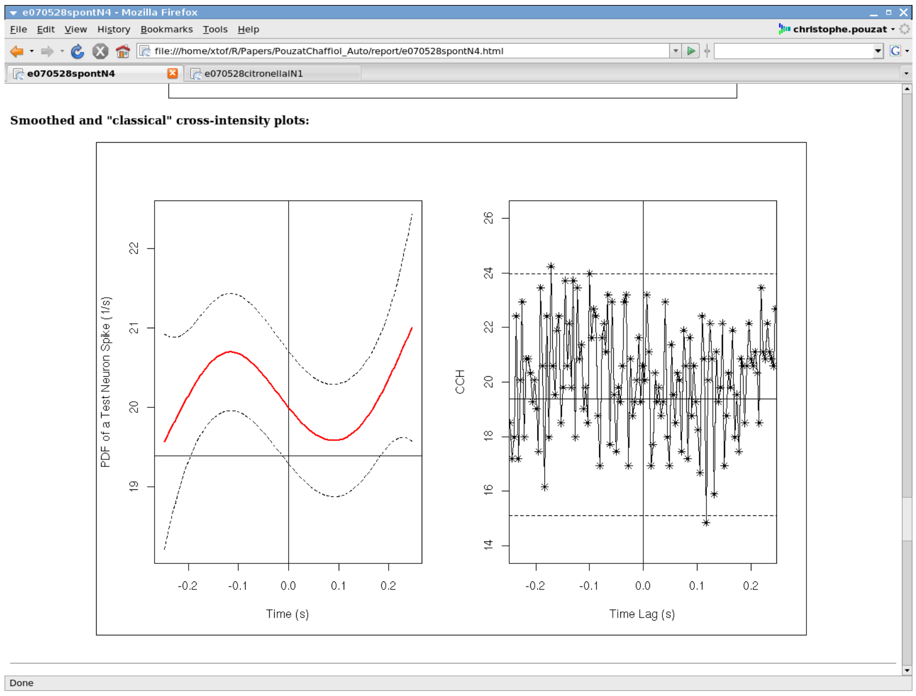
a spike in neuron 4 seems to be significant. But better tests, that will be explained in a subsequent paper, are needed before firm conclusions can be drawn.

Function reportHTML.spikeTrain also writes to disk a data file (using the R data format) where analysis results are stored. The data analyst can therefore quickly go back to the intermediate results if something appears suspicious in the report.

3.2 Stimulus response analysis

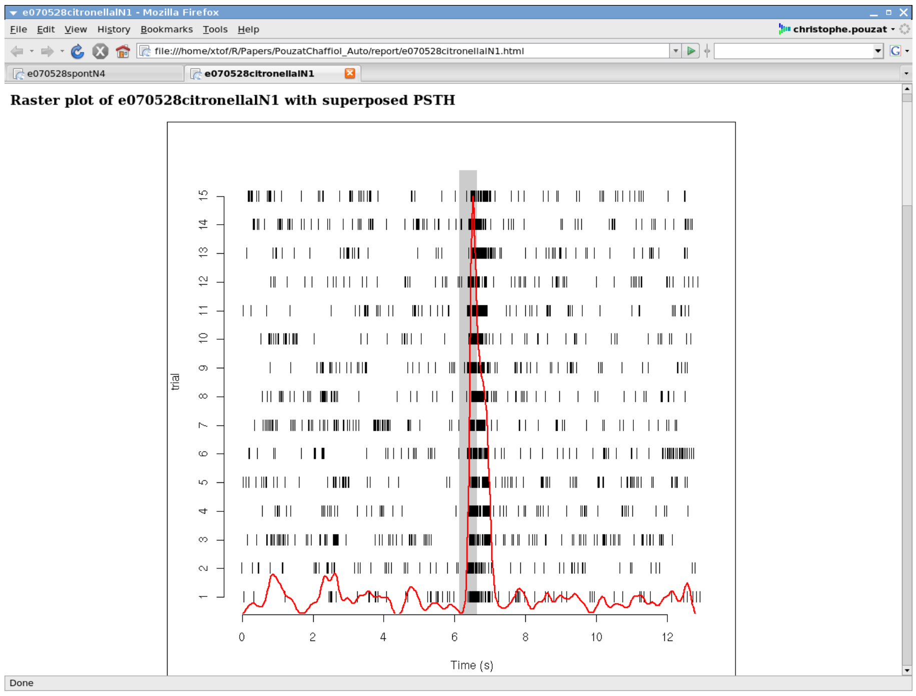
Like in Sec. 3.1, only few screen shots of the full report are shown. The full report can be seen in our web site[16] or, even better, can be generated by the reader as explained in Sec. A.19. Our automatic analysis and its report are now briefly described. A raster plot is added first to the report (Sec. 2.5.1) and a *smooth peri stimulus time histogram* (Sec. 2.5.2) is superposed to it (see Sec. A.18 for details). Fig. 9 shows a screen shot with this raster plot. The summary of the inhomogeneous Poisson fit (Sec. 2.5.2) leading the *spsth* is added next together with a short summary describing how accurate the hypothesis of constant intensity/rate made during the preprocessing was in view of the estimated rate (Fig. 10). A plot of the smooth PSTH with approximate 95% CI is added (not shown on screen shot but it looks like the right graph of Fig. 15). If argument doGamCheck is set to TRUE a diagnostic plot for the fitted inhomogeneous Poisson model is added (Sec. A.18 and Fig. 16). If argument doTimeTransformation is set to TRUE the estimated integrated intensity is used to perform a time transformation (Sec. 2.4.5) and Ogata's test plots are generated as illustrated on Fig. 11. It is clear from the latter that altough the *inhomogenous Poisson process* is a good model for the aggregated response, it is not a good model for the individual ones.

[16]http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR_folder/e070528citronellalN1.html


e070528spontN4 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
christophe.pouzat
file:///home/xtof/R/Papers/PouzatChaffiol_Auto/report/e070528spontN4.html
e070528spontN4
e070528citronellalN1
PDF of a Test Neuron Spike (1/s)
Time (s)
CCH
Time Lag (s)
Done

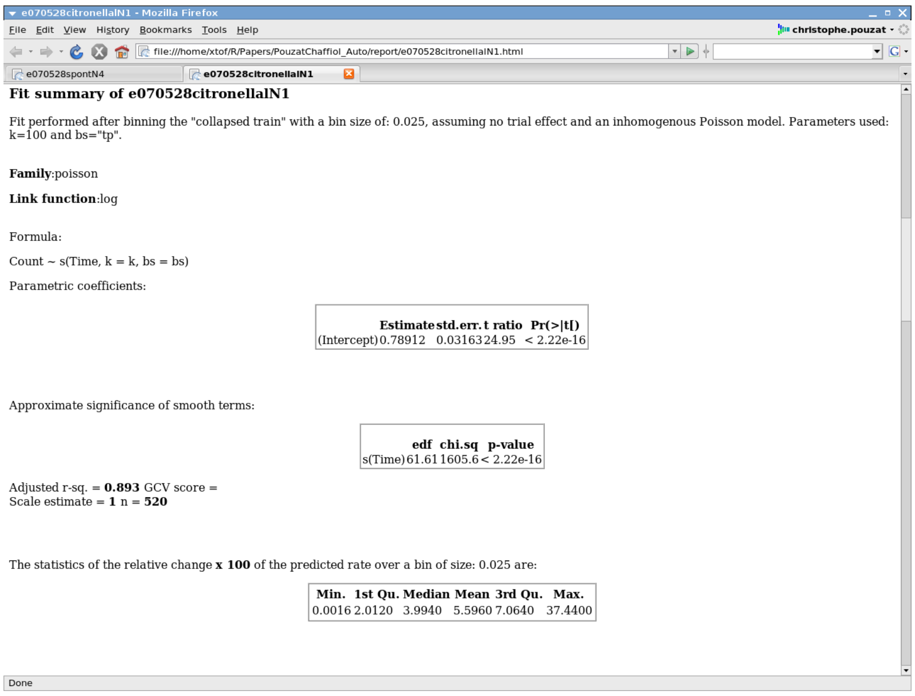
Smoothed and "classical" cross-intensity plots:

Figure 8: *Smooth* and "*classical*" *cross-correlation histograms* between neuron 4 (reference) and neuron 2 (test) as they appear in the automatically generated HTML report. One both plots the dotted lines define a pointwise 95 % confidence region.


e070528citronellalN1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
christophe.pouzat
file:///home/xtof/R/Papers/PouzatChaffiol_Auto/report/e070528citronellalN1.html
e070528spontN4
e070528citronellalN1
trial
Time (s)
Done

Raster plot of e070528citronellalN1 with superposed PSTH

Figure 9: Raster plot with superposed *smooth peri stimulus time histogram* (red curve) for neuron 1 as it appears in the automatically generated HTML report.


e070528citronellalN1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
christophe.pouzat
file:///home/xtof/R/Papers/PouzatChaffiol_Auto/report/e070528citronellalN1.html
e070528spontN4
e070528citronellalN1
Done

Fit summary of e070528citronellalN1

Fit performed after binning the "collapsed train" with a bin size of: 0.025, assuming no trial effect and an inhomogenous Poisson model. Parameters used: k=100 and bs="tp".

Family:poisson

Link function:log

Formula:

Count ~ s(Time, k = k, bs = bs)

Parametric coefficients:

	Estimate	std.err.	t ratio	Pr(>|t[)
(Intercept)	0.78912	0.03163	24.95	< 2.22e-16

Approximate significance of smooth terms:

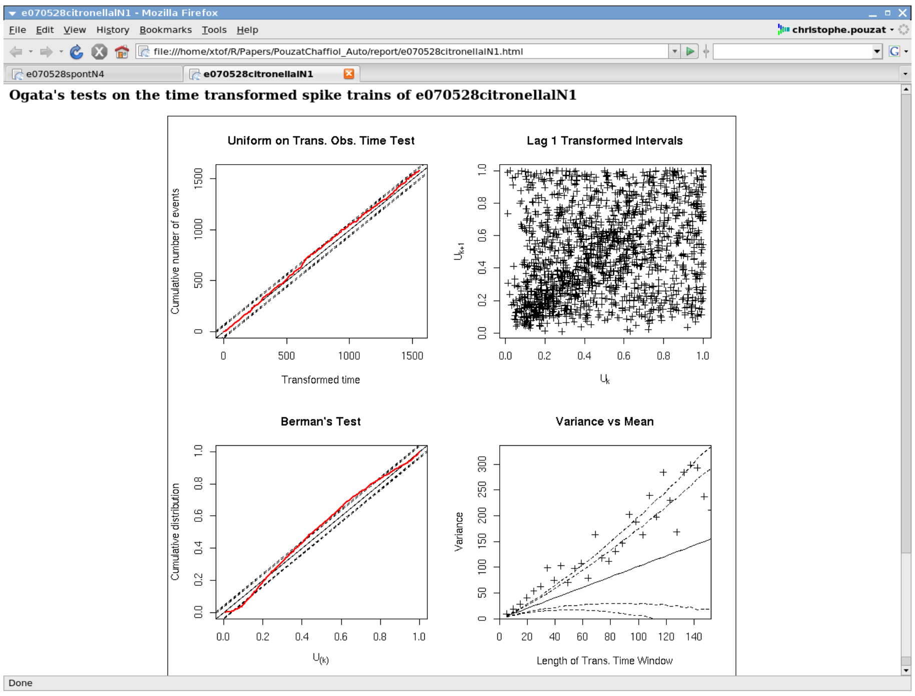
	edf	chi.sq	p-value
s(Time)	61.61	1605.6	< 2.22e-16

Adjusted r-sq. = **0.893** GCV score =
Scale estimate = **1** n = **520**

The statistics of the relative change **x 100** of the predicted rate over a bin of size: 0.025 are:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.0016	2.0120	3.9940	5.5960	7.0640	37.4400

Figure 10: Summary of the gam fit for neuron 1 as it appears in the automatically generated HTML report.



Ogata's tests on the time transformed spike trains of e070528citronellalN1

Figure 11: Event if the *inhomogenous Poisson process* is a good model for the aggregated response (Fig. 16) it is *not* an adequate model for the individual responses as shown by the Ogata's tests battery applied to the time transformed responses. Screen shot from the automatically generated HTML report.

4 Discussion

Experimental techniques used in modern neurocience research, like MEA recordings, tend to generate vast amounts of data. Experience moreover shows that the analysis of these raw data generates also a lot of "secondary" data. These quantitative aspects represent first a serious time challenge simply because data analysis requires time. Our answer to that challenge is to make the computer work for us. But remembering what John Tukey said: "Numerical quantities focus on expected values, graphical summaries on unexpected values", we make our computer not only compute but also generate a lot of diagnostic plots. Our approach is therefore to implement an automatic "robust" preliminary spike train analysis. Keeping in mind that real data have a tendency to wander out of our preset frames "we" generate and save many plot allowing us to scrutinize our analysis results, judge their trustworthiness and if necessary go back to specific stages of our analysis.

A second issue neurophysiologists have to face when dealing with MEA data analysis is the management of the "secondary" data they produce. Keeping things organized and easy to retrieve can become a problem especially for people who do not want to print 20 graphs per analyzed data set and/or have to move around with 20 kg of folders. The computer is again the solution when combined with the HTML file format. With this approach the 20 kg physical folder becomes an icon in the directory arborization of one's home directory. The analysis results become also easy to retrieve and can even be directly be shown to colleagues in lab meetings, as we hope our few screen shots have convinced our reader.

We have implemented both the "robust spike train analysis with lots of plots" approach and the HTML report generation in an open source and free package: STAR. Being open source our approach can easily be tailored to users/data specific needs. For our data it turns out to work "well" (in the sense of actually doing what we expect it to do) and fast (the run times reported in Sec. A.19 are typical as can be checked by the reader). But we insist again on the preliminary aspect of this automatic analysis. Most of the data we showed require more sophisticated analysis techniques in order to pass the Ogata's tests battery. But this will be the subject of another paper.

In conclusion we don't claim to have brought anything original from a methodological view point in this paper, but we hope that the bits we collected from our colleague works and the way we organized them in the STAR package will make spike train analysis more enjoyable to some.

A Reproducing the analysis/figures/report of this paper: A STAR tutorial

A.1 Getting R and STAR

R is an open source and free software that can be downloaded from: http://www.r-project.org or from any mirror site of the Comprehensive R Archive Network (CRAN). The software documentation and user contributed add-on packages can also be dowloaded from these sites. Precise instructions can be found on how to compile or just install from binary files the software on many different systems.

New users can get started by reading through: "An Introduction to R" which comes with the software[17]. Windows users who would feel a bit lost with the sober graphical user interface which comes with the Windows version should consider also installing "SciViews R GUI": http://www.sciviews.org/SciViews-R/.

STAR is not yet posted on CRAN (but it will be soon). It can be download from our web site: http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR.html. Once the proper version has been downloaded (Linux, which also runs on Mac, or Windows), it is installed like any other R add-on package. Check the documentation of function install.packages to see how to do it.

A.2 Preliminary remarks

We present in this appendix the full sequence of commands of R and STAR leading to the material, analysis, figures, etc, presented in this paper. We also try to give some background information on R, to help motivated users getting started. Here different fonts are going to be used to distinguish between what the user types following the R prompt which will appear like:

```
this is what an R input is going to look like in the sequel
```

and what R returns which will appear like:

```
this is how R outputs will appear
```

A.2.1 Getting help

Once R is started help on any command like plot can be obtained by typing the command name with ? as a prefix, *i.e.*:

```
> ?plot
```

It can also be done by using function help:

```
> help(plot)
```

A.2.2 Ending an R session

An R session is ended (from the command line) by the command:

```
> q()
```

[17]And that can also be found at: http://cran.r-project.org/doc/manuals/R-intro.html. In addition, among the user contributed documentations, we particularly like Emmanuel Paradis: "R for Beginners" (http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf) and Thomas Lumley: "R fundamentals" (http://faculty.washington.edu/tlumley/Rcourse/).

A.3 Loading STAR

To use STAR, an add-on package of R, the user has to load it into R search path with the library function as follows:

```
> library(STAR)
```

The effect of this command is to make STAR functions available to the user. Technically it adds a new environment into which the R interpreter looks for a variable/function name when a command is typed in by the user. Function search shows the search path used by the interpreter:

```
> search()

 [1] ".GlobalEnv"        "package:STAR"
 [3] "package:R2HTML"    "package:mgcv"
 [5] "package:sound"     "package:survival"
 [7] "package:splines"   "package:stats"
 [9] "package:graphics"  "package:grDevices"
[11] "package:utils"     "package:datasets"
[13] "package:methods"   "Autoloads"
[15] "package:base"
```

A.4 A comment on functions arguments

R functions like, library, often accept many arguments but only a few of them have to be *explicitly* specified by the user. The idea is to have both *control* over the function behavior and *ease of use*, *i.e.*, short command lines to type in. This is implemented by defining default values for the arguments, either explicitely in the argument list of the function or internally in the function's body. We can for instance look at the arguments accepted by function library by using function args with library as argument:

```
> args(library)

function (package, help, pos = 2, lib.loc = NULL, character.only = FALSE,
    logical.return = FALSE, warn.conflicts = TRUE,
    keep.source = getOption("keep.source.pkgs"),
    verbose = getOption("verbose"), version)
NULL
```

We see that among the 10 possible arguments of library the specification of a single one, package, was sufficient to do what we wanted. Using explicitely the other arguments with values different from their default ones would have allowed us to have a finer control on what the function does.

We also remark that when we entered args(library) we passed a function, library, as an argument of another function, args, without doing anything special. R being based on the Scheme langage [14] *does not* make any differences between functions and other types of objects.

A.5 Loading data

All the data sets used in this paper are part of the STAR data sets which means they can be loaded with function data. To load the data of the experiment of May 5 2007 in the spontaneous regime which is named: "e070528spont", we enter:

```
> data(e070528spont)
```

Clearly users will want to use STAR with their own data so we give a very brief description on how data can be loaded into R. We will load the spike train of the first neuron of the e070528spont data set. The data are in ASCII format on a distant machine and we are going to load them through a web connection. The data file e070528spontN1.txt is located on our lab server at the following address: http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR_folder/e070528spontN1.txt. The first four lines of the file give some information about the data:

```
Data set: e070528
Neuron: 1
Condition: spontaneous activity
By: Antoine Chaffiol
```

The following line is blank and the next 336 lines contain the spike times. We are going to use function scan to load the data. For editing purposes, we will moreover build the address of our file piece by piece: myURL will contain the url address of the folder containing the data file:

```
> myURL <- "http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR_folder"
```

and myFileName will be the name of the file per se:

```
> myFileName <- "e070528spontN1.txt"
```

Notice the use of, "<-", for assignments. The more common symbol, "=", could have also been used, *e.g.*:

```
> myFileName = "e070528spontN1.txt"
```

would have given the same result. The file address is then obtained by gluing together the 2 pieces, myURL and myFileName, with a "/" in between using function paste:

```
> myFullName <- paste(myURL, "/", myFileName, sep = "")
```

Function scan finishes the job and reads the data into our work space:

```
> e070528spontN1 <- scan(myFullName, skip = 5)
```

Notice that we used argument, skip, to start reading the file from the sixth line. If the data had been on our hard drive in the current working directory (see ?getwd and ?setwd) of R, we would have used:

```
> e070528spontN1 <- scan("e070528spontN1.txt", skip = 5)
```

The first argument of scan is just the "full path" to the data file, it does not matter if the path includes an internet connection or not. Functions are also available to read data in binary format (readBin) or with table structures (read.table). Check the R Data Import/Export manual which comes with the software for a comprehensive description of the data import/export capabilities of R.

The new object we have loaded into our R workspace, e070528spontN1, is a trite vector of double precision numbers, that is, a numeric object for R. To convert it into a spikeTrain object that STAR processes in a particular way, we use function as.spikeTrain:

```
> e070528spontN1 <- as.spikeTrain(e070528spontN1)
```

Function as.spikeTrain is not doing much, it merely checks that its argument can be a proper spikeTrain object (its elements should be strictly increasing) and gives spikeTrain class attribute to the object it returns. This probably looks obscur at that stage, but it should become clearer after the presentation of the "class / method" mechanism (Sec. A.7 and A.9).

A.6 A Comment on list objects

If we look of the type of object we loaded into our work space with function data, by calling function class on e070528spont:

```
> class(e070528spont)
```

```
[1] "list"
```

we see that it is a list. list objects are composite objects whose components can be indexed. The different components of a list don't have to be of the same type (or class to use the proper terminology). list objects are a very convenient way to keep related objects together. The number of components of a list is returned by function length:

```
> length(e070528spont)
```

```
[1] 4
```

Components of list objects can have names (it usually easier for a human to remember a meaningful name than a number) which are returned by function names:

```
> names(e070528spont)
```

```
[1] "neuron 1" "neuron 2" "neuron 3" "neuron 4"
```

list components can be accessed either by their index or by their name, *i.e.*:

```
> e070528spont[[4]]
```

gives the same result as:

```
> e070528spont[["neuron 4"]]
```

When dealing with indexed objects like list objects it often happens that we want to perform the same computation on every component of the object. We could for instance want to see what is the class of e070528spont components. A function which will do this task and spare us the job of writing a for loop is sapply[18]:

```
> sapply(e070528spont, class)
```

```
    neuron 1     neuron 2     neuron 3     neuron 4 
"spikeTrain" "spikeTrain" "spikeTrain" "spikeTrain" 
```

In a similar way we could get the length of these spikeTrain components (if they have one):

```
> sapply(e070528spont, length)
```

```
neuron 1 neuron 2 neuron 3 neuron 4 
     336     1173     1834     1015 
```

So we have learned that e070528spont is a list object with four named components of class spikeTrain each one with a different length. Of course we could have gotten almost the same information by looking at the documentation of e070528spont, by typing: ?e070528spont.

[18]Remember that when Ris running you can always get help on functions, like sapply, by typing: ?sapply.

A.7 A comment on the class / method mechanism

If we look at the documentation of function as.spikeTrain (?as.spikeTrain), which creates spikeTrain objects we learn that: "A spikeTrain object is a numeric vector whose elements are strictly increasing (that is, something which can be interpreted as a sequence of times of successive events with no two events occurring at the same time)." At first sight creating a new type (class) of objects which are just classical numeric vectors with a "small" peculiarity (the successive elements must be strictly increasing) would suggest that we are "over doing it". This would be ignoring the gains we can obtain from the "class / method mechanism".

When we work regularly with a type of data which has a specific structure and that we interprete in a specific way, we quickly end up with a "standard" way of plotting as well as summarizing them numerically. If our favorite software provides a general function for plotting objects say, plot, to use the name of the R function doing this job, we then often end up writing a new function or a short script which uses plot with arguments which are specific to the type of data we are looking at. When we have reached this stage it is worth thinking of using the "class method mechanism" provided by R[19]. By doing so *we will transfer the task of finding the proper* plot *function from us to the* R *interpreter*. This mechanism works by allowing users/programmers to create new "tailored" functions for some so called generic functionwhich are very frequently used in data analysis, like plot, print or summary. These object specific functions are called "methods" and the objects to which they apply must have a "class". e070528spont[["neuron 4"]] is for instance an object of class spikeTrain and we have written a method, plot.spikeTrain, which does the job of generating a plot in a spikeTrain specific manner. If we then want to plot e070528spont[["neuron 4"]], we don't have to remember that it is a spikeTrain object and enter:

```
> plot.spikeTrain(e070528spont[["neuron 4"]])
```

but only:

```
> plot(e070528spont[["neuron 4"]])
```

When plot(x) is entered, the R interpreter looks for the class of x, say, xClass, then it looks for a method called: plot.xClass. If such a method exists then the excecuted command is in fact: plot.xClass(x), otherwise it is: plot.default(x).

What have just written probably looks a bit abstract to the reader who has never been exposed to such ideas. It can also look over complicated to programmers used to a software environment which does not offer this functionality. But it turns out to be an extremly efficient concept. With it "casual" users can plot spikeTrain objects and obtain immediatly a *meaningful* display without having to know everything about the structure of spikeTrain objects or to know all the details of the plot function. For the non-casual user it means a big time gain when using the software because commands are much shorter. To the programmer it means more work on a short term, but signifiant gains on the mid- to long-term because it generates a better software organization.

A.8 Spike train plot generation

By now we should have guessed that Fig. 1 is simply generated by entering:

```
> plot(e070528spont[[4]])
```

In turns out that we could have generated the same figure (except the title which would have been slightly less informative) by entering:

[19]And of course to switch to R if we are not already using it!

```
> e070528spont[[4]]
```

When the name of an single object (like e070528spont[[4]]) is typed in the command line before pressing the return key, the R interpreter calls function print meaning that a print method specific to the class of the object is looked for before the evaluation is carried out. In the above exemple that means that what is evaluated really is:

```
> print.spikeTrain(e070528spont[[4]])
```

But we defined print.spikeTrain to be the same as plot.spikeTrain meaning that a plot is generated just by typing a spikeTrain name at the command line before pressing return.

A.9 The class / method mechanism in action

We have just seen how to generate a plot for spikeTrain objects using transparently the plot.spikeTrain method. As an illustration of the usefullness of the class /method mechanism, the reader can try the following sequence of commands. First generate a plot of the spikeTrain object, e070528spontN1, we created after loading some "raw data" (Sec. A.5):

```
> plot(e070528spontN1)
```

Now, remove the spikeTrain class attribute from e070528spontN1 with function unclass:

```
> e070528spontN1 <- unclass(e070528spontN1)
```

and plot it again:

```
> plot(e070528spontN1)
```

It is now plotted as "trite" numeric object, although its data content did not change at all.

A.10 Renewal test plot

A renewal test plot of neuron 4 of the e070528spont data set is obtained by calling function renewalTestPlot:

```
> renewalTestPlot(e070528spont[[4]])
```

As mentioned in Sec.2.4.3, these plots are sensitive non-stationarity detectors as illustrated in the pkDataSet1 demo of STAR. It uses the sPK data set (?sPK) and is launched as follows (the first command displays a list of the demos available in STAR with a short description):

```
> demo(package = "STAR")
> demo(pkDataSet1)
```

A.11 Comparing duration distribution

The *theoretical quantile quantile plot*s of the six duration distributions fitted to the *isi*s of neuron 1 of the e070528spont data set are generated by function compModels, which also does the fits and returns the *Akaike's Information Criterion* value for each model.

```
> compModels(e070528spont[[1]])

 invgauss     lnorm    llogis   weibull      rexp     gamma
-594.6524 -573.2588 -556.4574 -505.4907 -501.7978 -489.7187
```

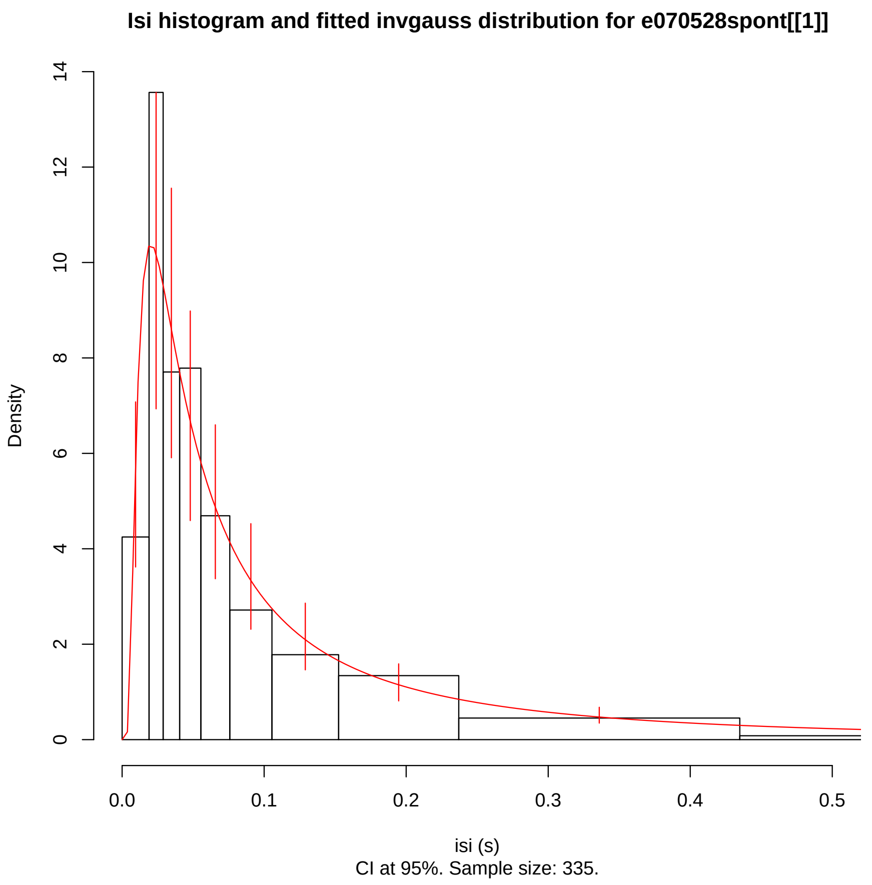


Figure 12: *isi* histogram (black rectangles) with superposed fitted inverse Gaussian density (red curve). The 10 histogram bins are set after the fit such that if the fit was good, 10% of the *isi*s would fall into each bin. Data of neuron 1, data set, e070528, spontaneous regime. For more details: ?isiHistFit.

A.12 Comparison with the *isi* histogram

Although we don't use *isi* histograms in our automatic spike train processing, STAR has a function to plot it together with the fit of one of the 6 duration distributions: isiHistFit. To superpose a fitted inverse Gaussian density, to the empirical *isi* histogram of neuron 1 of the e070528spont as shown on Fig. 12, we would enter:

```
> isiHistFit(e070528spont[[1]], "invgauss", xlim = c(0,
+     0.5))
```

A.13 Doing your own *isi* histogram generating function

We have not included any specific function to create a "simple" histogram from the *isi*s of a spikeTrain object. This is because the job is done easily by calling two functions successively. The *isi*s are obtained from a spikeTrain object, like e070528spont[[4]], by calling method diff without further arguments (see ?diff.spikeTrain). Then hist is called on the result. So we are now ready to create our first R function, isiHist4ST:

```
> isiHist4ST <- function(mySpikeTrain, ...) hist(diff(mySpikeTrain),
+     ...)
```

We could even go further and create a hist method for spikeTrain objects considering that only a histogram of the *isis* of the train would make sense:

```
> hist.spikeTrain <- function(mySpikeTrain, ...) hist(diff(mySpikeTrain),
+     ...)
```

Simple isn't it?

A.14 Walk through time transformation

We will here illustrate the time transformation of Sec. 2.4.5 with neuron 1 of the e070528spont. As we have seen in Sec. A.11, the best *homogenous renewal process* model for this neuron is the inverse Gaussian. So we start by getting the *maximum likelihood estimate* for the parameters of this distribution using function invgaussMLE:

```
> e070528spontN1.fit <- invgaussMLE(e070528spont[[1]])
```

We then perform the time transformation following the scheme of Eq. 18 and 19 using function pinvgauss (check ?pinvgauss):

```
> e070528spontN1.theta <- e070528spontN1.fit$estimate
> e070528spontN1.TT <- c(0, cumsum(-pinvgauss(diff(e070528spont[[1]]),
+     e070528spontN1.theta[1], e070528spontN1.theta[2],
+     log.p = TRUE, lower.tail = FALSE)))
```

Now we can plot the result as a composite plot, the *Counting process* and the transformed time as functions of actual spike times and then the *Counting process* as a function of transformed time:

```
> X <- unclass(e070528spont[[1]])
> layout(matrix(1:2, nrow = 1))
> plot(X, seq(X), type = "l", xlab = "Spike Times (s)",
+     ylab = "Cumulative Number of Events", lwd = 2)
> lines(X, e070528spontN1.TT, col = 2, lwd = 2)
> plot(e070528spontN1.TT, seq(X), type = "n", xlab = "Transformed Spike Times",
+     ylab = "", col = 2)
> abline(a = 0, b = 1, lty = 2)
> lines(e070528spontN1.TT, seq(X), col = 2, lwd = 2)
```

A.15 Ogata's tests examples

We show here how to generate the Ogata's test battery shown at the end of Sec. 2.4.5. They are simply obtained by first setting the class of the e070528spontN1.TT to transformedTrain before calling plot (*i.e.*, plot.transformedTrain, check, ?plot.transformedTrain) on the object:

```
> class(e070528spontN1.TT) <- c("transformedTrain",
+     "spikeTrain")
> plot(e070528spontN1.TT, which = c(1, 2, 4, 5),
+     ask = FALSE)
```

Notice that we also set the "second" class of e070528spontN1.TT to spikeTrain. That allows us to use directly function renewalTestPlot on the object.

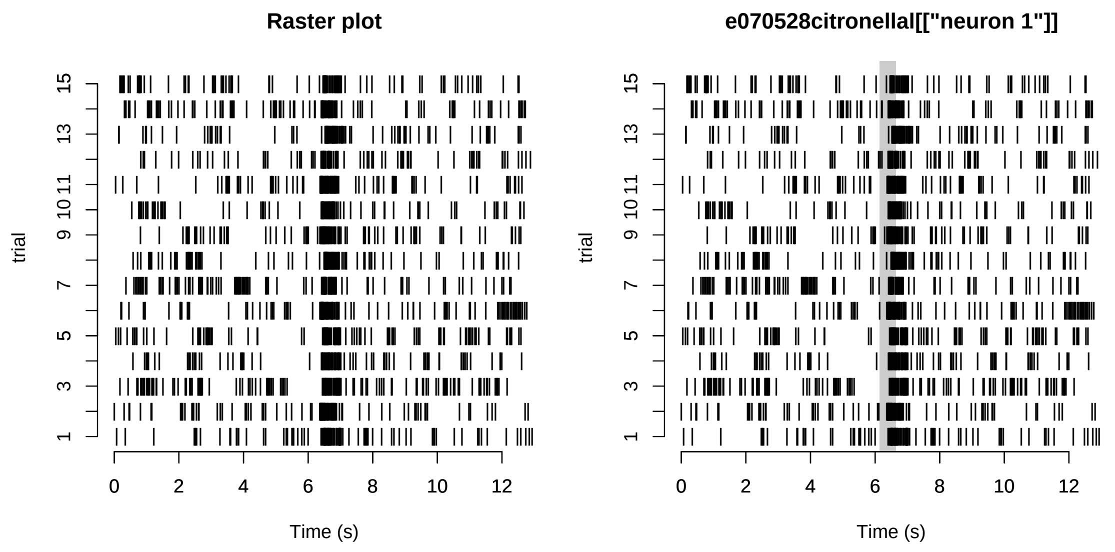


Figure 13: Illustration of print and plot methods for repeatedTrain objects. The 15 responses of neuron 1, data set e070528citronellal are used here. Left, plot generated by: e070528citronellal[["neuron 1"]]. Right, plot generated by: plot(e070528citronellal[["neuron 1"]],stim=c(6.14,6.64),main="e070528citronellal[[ñeuron 1]]"). Here the user has control over the plot title and argument stim (short for stimTimeCourse) controls the presence of the grey rectangle in the background (signalling the odor delivery).

A.16 The repeatedTrain class and associated methods

To illustrate the STAR functions for dealing with "stimulus responses" we are going to use the citronellal responses of the previous experiment. They are found under the name: e070528citronellal in STAR. 15 stimulations (0.5 s long) were applied with 1 minute intervals. As in Sec.A.5 we load the data into our work space with function data:

```
> data(e070528citronellal)
```

The data set documentation (?e070528citronellal) tells us that e070528citronellal is a list of repeatedTrain objects, which are themselves lists of spikeTrain objects (see ?as.repeatedTrain). Like in the spikeTrain case, the motivation to create a new class was to have methods specifically tailored to their objects. For instance the print method (print.repeatedTrain) generates a raster plot (Fig. 13 Left):

```
> e070528citronellal[["neuron 1"]]
```

while the plot method (plot.repeatedTrain) does the same while allowing for a better control of the output with, for instance, the specification of the stimTimeCourse argument to make the stimulus time course appear on the plot (Fig. 13 Right):

```
> plot(e070528citronellal[["neuron 1"]], stim = c(6.14,
+     6.64), main = "e070528citronellal[[\"neuron 1\"]]")
```

Here we have specified stim instead of stimTimeCourse since partial matching is used[20] when matching functions arguments.

[20]See Se. 4.3.2: Argument matching of *R Language Definition*. http://cran.r-project.org/doc/manuals/R-lang.html.

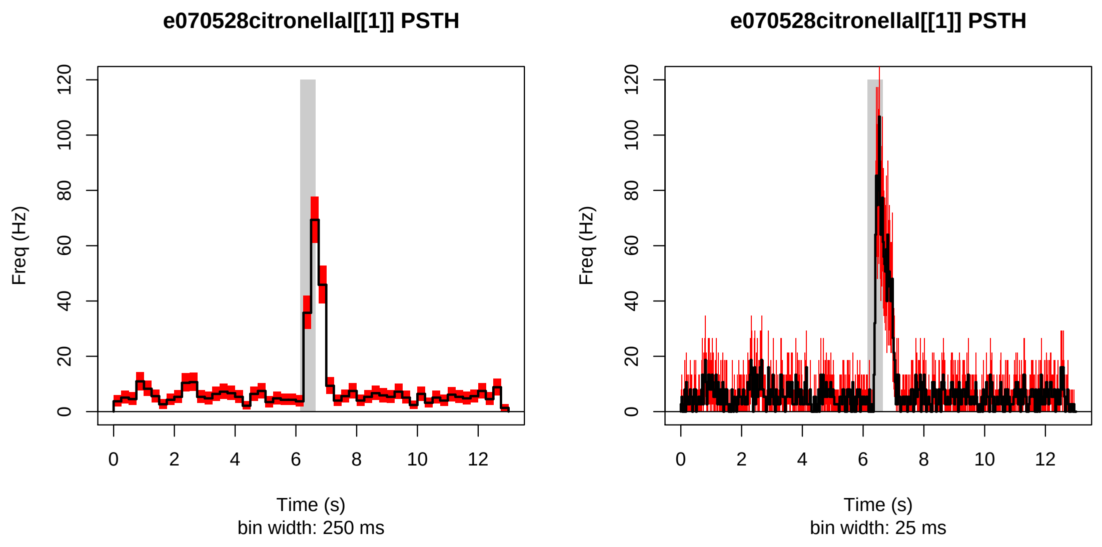


Figure 14: Illustration of psth function. The 15 responses of neuron 1, data set e070528citronellal are used here. Left, *psth* obtained with a bin width of 250 ms. Right, *psth* obtained with a bin width of 25 ms. The 95% confidence region appears in red.

A.17 Classical *psth*s with STAR

We have also included in STAR classes and methods for "classical" *psth*s. Function psth plots or returns a psth object. And method plot (plot.psth) plots it if it was not already done by psth. We will use them to illustrate the interactive bin width setting process described in Sec. 2.5.2. Using the same data as in the previous section we will construct 2 *psth*s, one with a bin width of 250 ms, the other one with a bin width of 25 ms. The plots (Fig. 14) also shows confidence intervals:

```
> psth(e070528citronellal[[1]], breaks = seq(0,
+     13, 0.25), colCI = 2, ylim = c(0, 120), sub = "bin width: 250 ms",
+     stim = c(6.14, 6.64))
> psth(e070528citronellal[[1]], breaks = seq(0,
+     13, 0.025), colCI = 2, ylim = c(0, 120), sub = "bin width: 25 ms",
+     stim = c(6.14, 6.64))
```

A.18 Details on *spsth*s

Smooth peri stimulus time histograms are obtained in STAR with the spsth function (see ?spsth). Compared to the construction of a *psth* , the preprocessing step involves a "too small" bin width. Here small refers to the fastest time constant expected to be present in the instantaneous firing rate. By default it is set to 25 ms. That can of course be changed by the user. Function spsth calls function gam of Simon Wood's package mgcv after this preprocessing. gam does the real work of getting the *spsth*. It uses *penalized regression splines* to get the smooth estimate. Several spline bases can be used like cubic splines but a very attractive feature of mgcv is that it includes *thin plate splines*. These splines *do not* require the positions of the knots to be fixed [31, Wood, 2006, pp 154-160] which eliminates one of the shortcomings of regression splines mentioned by [12].

Continuing with our previous example we would get a spsth object with:

```
> e070528citronN1.spsth.tp <- spsth(e070528citronellal[[1]],
+     plot = FALSE, bs = "tp")
```

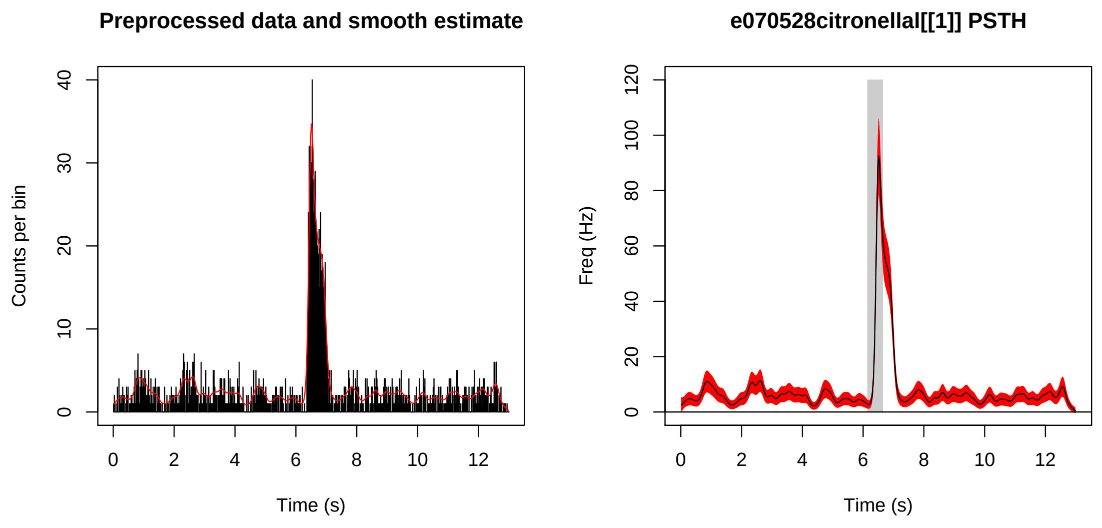


Figure 15: Illustration of spsth function. Same data as Fig. 14. Left, preprocessed data (black) used to obtain the "smooth" (red). Right, *spsth* obtained with function spsth. The 95% confidence region appears in red. The Y axis scale has been adjusted to facilitate comparison with Fig. 14.

It can be worth comparing at that stage the preprocessed data out of wich the "smooth" was constructed with the smooth itself. This is illustrated on Fig. 15 (Left) and obtained as follows:

```
> X <- e070528citronN1.spsth.tp$mids
> Counts <- e070528citronN1.spsth.tp$counts
> theBS <- diff(X)[1]
> nbTrials <- e070528citronN1.spsth.tp$nbTrials
> Y <- e070528citronN1.spsth.tp$lambdaFct(X) * theBS *
+     nbTrials

> plot(X, Counts, type = "h", xlab = "Time (s)",
+     ylab = "Counts per bin", main = "Preprocessed data and smooth estimate")
> lines(X, Y, col = 2, lwd = 1)
```

The plot of the actual *spsth* (Fig. 15, Right) is obtained with the plot method (plot.spsth):

```
> plot(e070528citronN1.spsth.tp, colCI = 2, lwd = 1,
+     stim = c(6.14, 6.64), ylim = c(0, 120))
```

When using gam it is safe to check that the maximal number of knots used was large enough. This maximal number of knots is given by argument k-1 of spsth and is set to 100 by default (that is the maximal number of knots is 99 by default). Function summary for spsth objects (that is, summary.spsth) returns information about the fitted model:

```
> summary(e070528citronN1.spsth.tp)

Family: poisson
Link function: log

Formula:
Count ~ s(Time, k = k, bs = bs)
```

```
Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.78912    0.03163   24.95   <2e-16 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Approximate significance of smooth terms:
          edf Est.rank Chi.sq p-value
s(Time) 61.61       97   1606  <2e-16 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

R-sq.(adj) =  0.893   Deviance explained = 77.5%
UBRE score = 0.15506  Scale est. = 1         n = 520
```

Users should check that the number under the edf (for equivalent degrees of freedom) heading is smaller than k-1. See ?choose.k for additional details.

Finally it is always a good idea to check that the fitted model gives a reasonable account of the data at hand. Diagnostic plots as well as numeric summaries for objects returned by function gam (see ?gamObject) are returned by function gam.check. The gam objects generated by the call to spsth can themselves be obtained with function gamObj of STAR:

```
> gam.check(gamObj(e070528citronN1.spsth.tp))
```

```
fit method: deviance based outer iter. - newton, exact hessian.
full convergence after 4 iterations.
gradient range [9.692206e-09,9.692206e-09] (score 0.1550627 & scale 1).
Hessian positive definite, eigenvalue range [0.008241519,0.008241519].
```

The plot is shown on Fig. 16. These diagnostic plots are direct generalizations of the plots used for generalized linear models in R [10]. They are fully explained in Simon Wood's book [31]. The upper left graph shows the model residuals quantiles against the quantiles of a normal distribution (which should also be the distribution of the residuals if the fit is good). The upper right show the residual against the "linear predictor" (in our case the log of the estimated instantaneous firing rate). Ideally points should be homogeneously scatter along the y axis (that is, the scatter should not depend on the x value). It is not strictly the case here because some of the bin counts generated by our preprocessing stage are very small which generates the banded structure. The lower left graph is a residual histogram which should look like a standard Gaussian if the null hypothesis is correct. The lower right graph shows the response versus the fitted value and should look like a straight line going through the origin with a slope 1.

A.19 Automatic analysis and report generation

The screen shots making the figures of the "Results" section are obtained by calling method reportHTMLon a spikeTrain object (reportHTML.spikeTrain) and on a repeatedTrain object (reportHTML.repeatedTrain). Let us start with the former, assuming we have already created a "report" subdirectory in our working directory:

```
> reportHTML(object = e070528spont[["neuron 4"]],
+     filename = "e070528spontN4", otherST = e070528spont[-4],
+     laglim = c(-1, 1) * 0.25, forceTT = TRUE,
+     directory = "report")
```

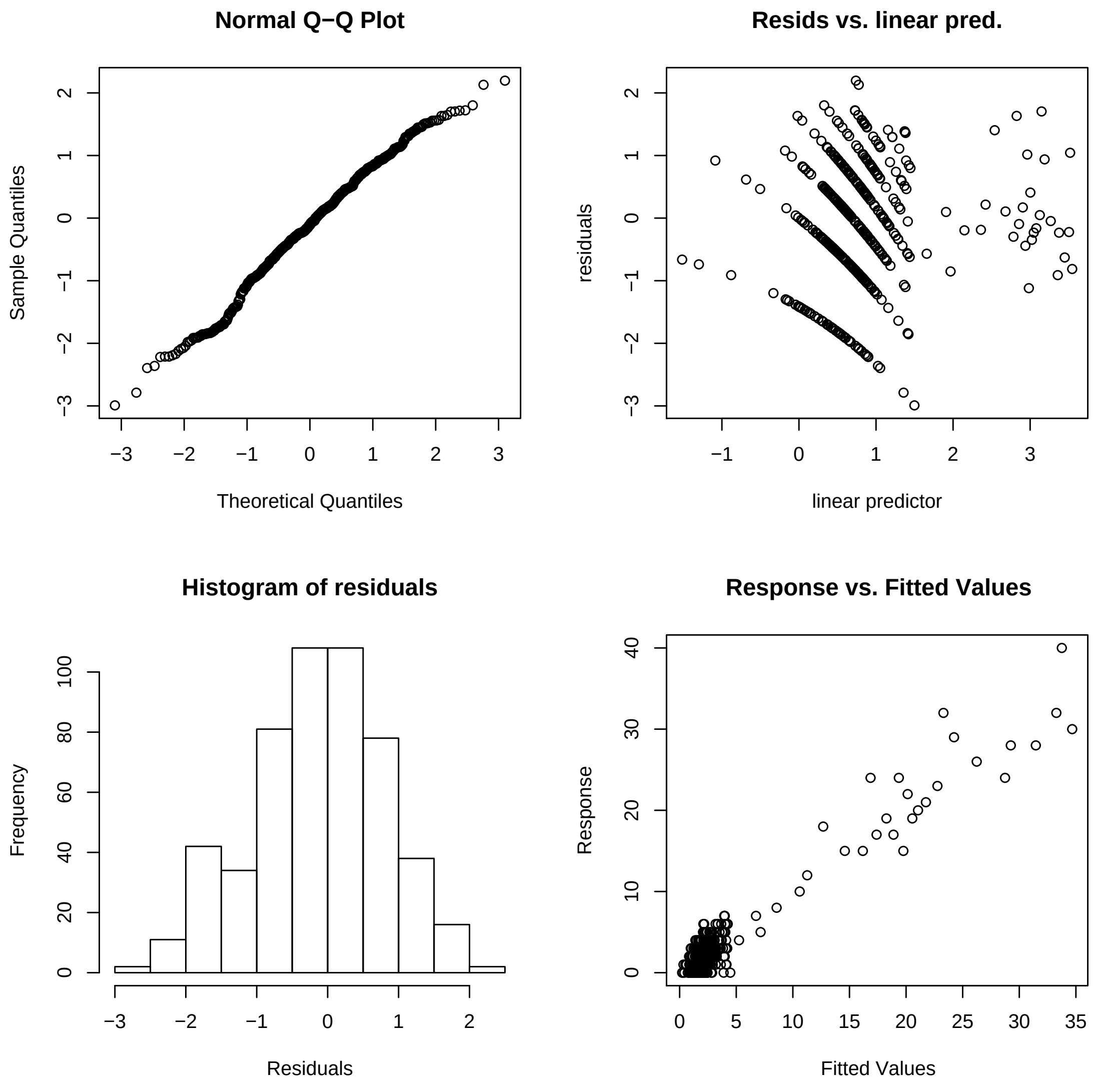


Figure 16: Plot generated by applying function gam.check to e070528citronN1.spsth.tp (via a call of gamObj). See text for details.

This takes about 30 s on a laptop equiped with a Core2 CPU at 2 GHz. Argument forceTT controls the generation of the *theoretical quantile quantile plots* and of the Ogata's tests battery. If is is set to FALSE the plots and tests are included in the report only if one of the 6 duration distribution models fits the data. Passing lists of spike trains via argument otherST induces the generation of both types of *cross-correlation histograms* (by default). The lag of these *cross-correlation histograms* is controled by argument laglim. For more details, see ?reportHTML.spikeTrain.

The report of a repeatedTrain object illustrated in Sec. 3.2 is obtained with:

```
> reportHTML(object = e070528citronellal[["neuron 1"]],
+     filename = "e070528citronellalN1", stim = c(6.14,
+         6.64), directory = "report")
```

This takes about 56 s on a laptop equiped with a Core2 CPU at 2 GHz.

A.20 Software versions used for this tutorial

The versions of R and of the other packages used in this tutorial are obtained with function sessionInfo:

```
> sessionInfo()

R version 2.6.0 (2007-10-03)
i686-pc-linux-gnu

locale:
LC_CTYPE=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=en_US.UTF-8;LC_COLLATE=en_US.UTF-8;LC_MONETARY=

attached base packages:
[1] splines   stats     graphics  grDevices utils
[6] datasets  methods   base

other attached packages:
[1] STAR_0.1-5    R2HTML_1.58   mgcv_1.3-28   sound_1.1
[5] survival_2.34

loaded via a namespace (and not attached):
[1] tools_2.6.0
```

References

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, second edition edition, 1996. Book website:http://mitpress.mit.edu/sicp/full-text/book/book.html.

[2] D. M. Bates and D. G. Watts. *Nonlinear Regression Analysis and Its Applications*. Wiley, 1988.

[3] D. R. Brillinger. Maximum likelihood analysis of spike trains of interacting nerve cells. *Biol Cybern*, 59(3):189–200, 1988.

[4] D. R. Brillinger, H. L. Bryant, and J. P. Segundo. Identification of synaptic interactions. *Biol Cybern*, 22(4):213–228, May 1976.

[5] Emery N Brown, Riccardo Barbieri, Valérie Ventura, Robert E Kass, and Loren M Frank. The time-rescaling theorem and its application to neural spike train data analysis. *Neural Comput*, 14(2):325–346, Feb 2002. Available from: http://www.stat.cmu.edu/~kass/papers/rescaling.pdf.

[6] Kenneth P. Burnham and David R. Anderson. *MODEL SLECTION AND MULTIMODEL INFERENCE. A Practical Information-Theoretic Approach*. Springer, 2nd edition, 2002.

[7] John Chambers. Computing with Data: Concepts and Challenges. *The American Statistician*, 53(1):73–84, 1999. Available from: http://cm.bell-labs.com/stat/doc/Neyman98.ps.

[8] John M. Chambers. Users, programmers and statistical software. *Journal of Computational and Graphical Statistics*, 9(3):404–422, sep 2000. Available from: http://cm.bell-labs.com/stat/doc/jmcJCGS2000.ps.

[9] John M. Chambers, William S. Cleveland, Beat Kleiner, and Paul A. Tukey. *GRAPHICAL METHODS FOR DATA ANALYSIS*. Wadsworth & Brooks/Cole, 1983.

[10] John M. Chambers and Trevor J. Hastie. *STATISTICAL MODELS IN S*. Chapman & Hall / CRC, 1992.

[11] D. R. Cox and P. A. W. Lewis. *The Statistical Analysis of Series of Events*. John Wiley & Sons, 1966.

[12] I. Dimatteo, C.R. Genovese, and R.E. Kass. Bayesian curve-fitting with free-knot splines. *Biometrika*, 88:1055–1071, 2001. Available from: http://www.stat.cmu.edu/~kass/papers/bars.pdf.

[13] George L. Gerstein and Nelson Y.-S. Kiang. An approach to the quantitative analysis of electrophysiological data from single neurons. *Biophysical Journal*, 1(1):15–28, September 1960. Available form: http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed&pubmedid=13704760.

[14] R Ihaka and R Gentleman. R: A Language for Data Analysis and Graphics. *Journal of Graphical and Computational Statistics*, 5:299–314, 1996.

[15] D.H. Johnson. Point process models of single-neuron discharges. *J. Computational Neuroscience*, 3(4):275–299, 1996.

[16] J. G. Kalbfleisch. *Probability and Statistical Inference. Volume 2: Statistical Inference*. Springer Texts in Statistics. Springer-Verlag, second edition, 1985.

[17] Robert E Kass, Valérie Ventura, and Can Cai. Statistical smoothing of neuronal data. *Network: Computation in Neural Systems*, 14(1):5–15, 2003. Available from: http://www.stat.cmu.edu/~kass/papers/smooth.pdf.

[18] Eric Lecoutre. The R2HTML package. *R News*, 3(3):33–36, December 2003. Available from: http://cran.r-project.org/doc/Rnews/Rnews_2003-3.pdf.

[19] J.K. Lindsey. *Introduction to Applied Statistics: A Modelling Approach.* Oxford University Press, 2004.

[20] John F. Monahan. *Numerical Methods of Statistics.* Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, first edition, 2001.

[21] Yosihiko Ogata. Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association*, 83(401):9–27, 1988.

[22] Didier Pelat. Bruits et Signaux. Doctoral school lectures notes (in French). Available from:http://cel.archives-ouvertes.fr/cel-00092937/fr/, 1996.

[23] D. H. Perkel, G. L. Gerstein, and G. P. Moore. Neuronal spike trains and stochastic point processes. I the single spike train. *Biophys. J.*, 7:391–418, 1967. Available from: http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed&pubmedid=4292791.

[24] D. H. Perkel, G. L. Gerstein, and G. P. Moore. Neuronal spike trains and stochastic point processes. II simultaneous spike trains. *Biophys. J.*, 7:419–440, 1967. Available from: http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed&pubmedid=4292792.

[25] Christophe Pouzat. *The New SpikeOMatic Tutorial*, 2006. Available from: http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/Data_folder/newSOMtutorial.pdf.

[26] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[27] David Ruppert, M. P. Wand, and R. J. Carroll. *Semiparametric Regression.* Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2003.

[28] Lon Turnbull, Emese Dian, and Guenter Gross. The string method of burst identification in neuronal spike trains. *J Neurosci Methods*, 145(1-2):23–35, Jun 2005.

[29] Valerie Ventura, Roberto Carta, Robert E. Kass, Sonya N. Gettner, and Carl R. Olson. Statistical analysis of temporal evolution in single-neuron firing rates. *Biostat*, 3(1):1–20, 2002. Available from: http://www.stat.cmu.edu/~kass/papers/temporal.pdf.

[30] Simon N. Wood. mgcv: GAMs and generalized ridge regression for R. *R News*, 1(2):20–25, June 2001. Available from: http://cran.r-project.org/doc/Rnews/Rnews_2001-2.pdf.

[31] Simon N. Wood. *Generalized Additive Models. An Introduction with R.* Chapman & Hall/CRC, 2006.

List of Figures

Index