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

September 8, 2008

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 a mixture of common and original procedures 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.

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* [10,43].
- 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* [38, 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 *smoothing spline* [46, 18, 19] 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] [40] 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` [21, 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].
- It is specifically designed to implement a clear and thorough type of data analysis [6].

[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 very easily extended by its users and, so to speak, encourages its users to become its programmers [7, 8].

Adopting R did moreover provide a pretty straightforward solution to our "analysis results organization and display" problem. Modern computers are all equipped 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 reasonable 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] [27], which turns out to be very simple to use.

Equipped 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 95 functions, most of which are seldom *directly* used. It has now reached a satisfying maturity when applied to our data recorded from the antennal 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, recordings and data sets

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 (as shown on Fig. 2 of [14]). 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.

2.1.4 Data sets

The examples used in this paper come from 4 experiments referred to as: `e060517`, `e060817`, `e060824` and `e070528` (the names are built with 2 digits for the year, two for the month and 2 for the day). The number of neurons reliably isolated in each of these experiments were respectively: 3, 3, 2 and 4. The spike trains used in this paper, either from spontaneous activity or from repeated odor stimulations are all distributed with our package STAR. For each of these experiments the spontaneous activity of each neuron recorded for $\sim$ 1 mn (58 to 61 s) is used. These specific data are referred to as `eNUMBER-spont_NEURON` in Sec. 3.1 and Table 1. One odor application with ionon (19 repetitions, 15 s acquired per presentation) is used for `e060517` and the corresponding data set is referred to as: `e060517ionon`. Three odor applications with citronellal, terpineol and a 50 / 50 mixture of the two (20 repetitions, 15 s acquired per presentation) are used for `e060817` and the corresponding data sets are refered to as: `e060817citron`, `e060817terpi` and `e060817mix`. One odor application with citral (20 repetitions, 15 s acquired per presentation) is used for `e060824` and the corresponding data set is referred to as: `e060824citral`. One odor application with citronellal (15 repetitions, 13 s acquired per presentation) is used for `e070528` and the corresponding data set is referred to as: `e070528citronellal`. A more comprehensive description of these data sets can be found in the help files of STAR.

2.2 Data analysis

All the data analysis described in this paper was carried out using R [40], 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 description 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" [39].

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

[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/

horizontal coordinate and where a symbol like a little vertical bar, or a star ("*") is used to represent each spike. As abundantly illustrated by [43, Turnbull et al, 2005] and, in fact, proposed much earlier in the first figure of the book of [10, 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 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, 22]. 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.

2.4.2 Poisson process

In the simplest case neuronal discharges are be well approximated by a *homogenous Poisson process*, which is formally defined as a stochastic process fully characterized by a *time independent rate parameter*, ν, such that the number of events observed in $(t, t+\tau]$ follows a *Poisson distribution* with parameter $\nu\tau$ (for all $t > 0$). Using our previous *Counting process* definition (Eq. 1) we would write:

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

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 its extension to time dependent discharges (with the *inhomogenous Poisson process*) is extremely useful to describe mean responses of a single neuron to repeated presentations of a given stimulus.

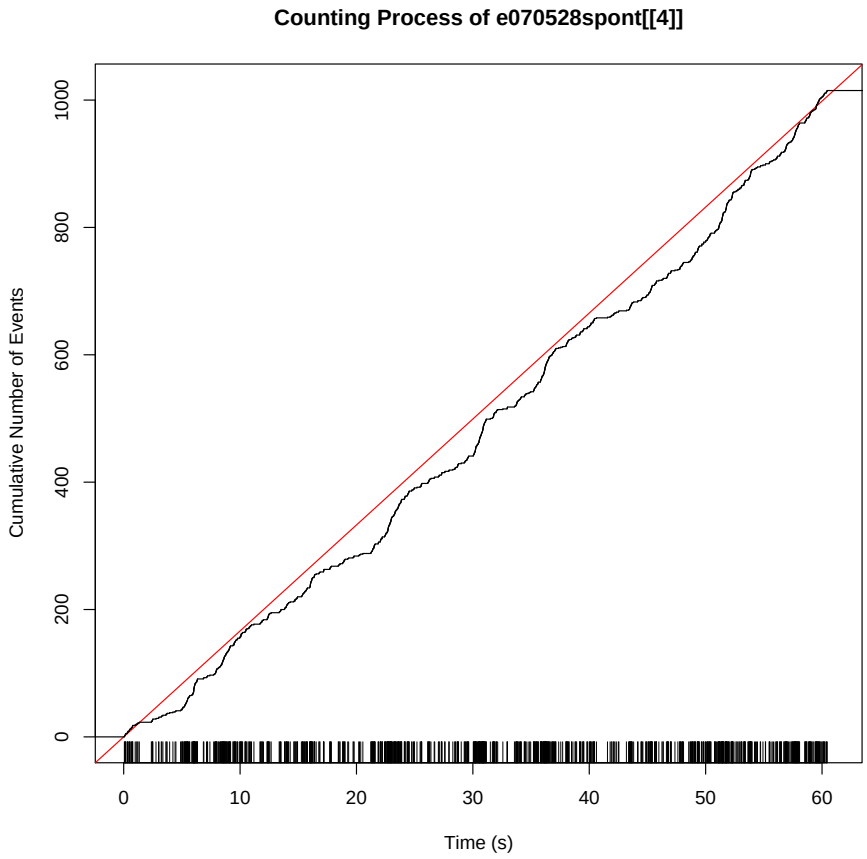

Figure 1: A spike train plot of neuron 4, data set, e070528, spontaneous regime. The generation of this figure with STAR is explained in Sec. A.8.

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 homogeneous renewal process is a process where the intervals between successive events, the *inter spike intervals* (*isi*), 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 original (unsorted sequence) in the new sorted one. To use a concrete example, let us assume 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 $\{O_j\}$ sequence is:

```
1 2 3 4 5
1 3 4 5 2
```

and the corresponding sorted sequence, $\{i_{(j)}\}$, is:

```
    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 a 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 as shown on Fig. 13. We can then subdivide the surface defined by the $\{1,\ldots,K-1\} \times \{1,\ldots,K-1\}$ square into sub-squares 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} vs O_j and O_{j+2} 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 [37, 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.

[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

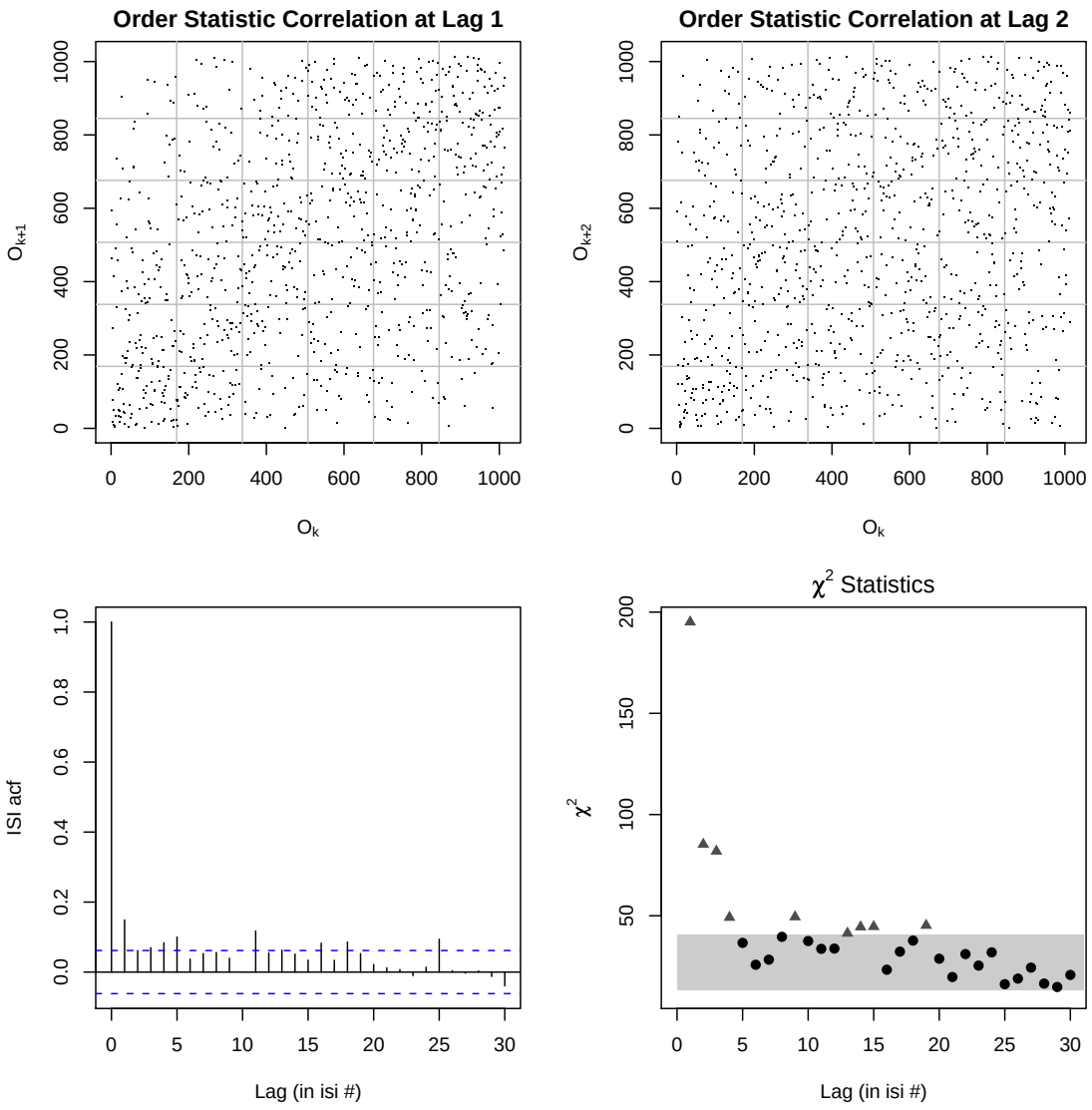

Figure 2: A "renewalTestPlot" of neuron 4, data set, e070528, spontaneous regime. The generation of this figure with STAR is explained in Sec. A.10.

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 [31] whose *probability density functions* (*pdf*s) are:

- log normal:

$$\text{dlnorm}(i;\mu,\sigma^2) = \frac{1}{\sqrt{i2\pi\sigma^2}} \exp\big(-\frac{1}{2}\frac{(\log i-\mu)^2}{\sigma^2}\big) \tag{3}$$

- inverse Gaussian:

$$\text{dinvgauss}(i;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2 i^3}} \exp\big(-\frac{1}{2}\frac{(i-\mu)^2}{i\sigma^2\mu^2}\big) \tag{4}$$

- gamma:

$$\text{dgamma}(i;\alpha,\beta) = \frac{i^{\alpha-1}}{\beta^\alpha\Gamma(\alpha)} \exp(-\frac{i}{\alpha}) \tag{5}$$

- Weibull:

$$\text{dweibull}(i;\alpha,\beta) = \frac{\alpha}{\beta}(\frac{i}{\beta})^{\alpha-1} \exp(-(\frac{i}{\beta})^\alpha) \tag{6}$$

- refractory exponential:

$$\text{drexp}(i;\alpha,i_{min}) = \begin{cases} 0 & \text{if } i < i_{min} \\ \alpha\, \exp(-\alpha\,(i-i_{min})) & \text{if } i \geq i_{min} \end{cases} \tag{7}$$

- log logistic:

$$\text{dllogis}(i;\mu,\sigma) = \frac{1}{i\sigma}\frac{\exp(-\frac{\log i-\mu}{\sigma})}{\big(1+\exp(-\frac{\log i-\mu}{\sigma})\big)^2} \tag{8}$$

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 [24]. The ones of the lognormal, inverse Gaussian and refractory exponential distributions are available in closed form [31]. 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 [32, Monahan, 2001, pp 210-216]. Initial guesses are obtained by the method of moments. In addition reparametrization is used systematically for parameters which are constrained to be positive [2, 32], 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 capacities of the models to fit the data are compared with the *Akaike's Information Criterion*[11] [5,31]. 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* (Q-Q plots) [9, Chambers et al, 1983, Chap. 6] as shown on Fig. A.12. 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[12].

[11] Strictly speaking, the *Akaike's Information Criterion* being designed to compare models with different number of parameters is not required here, a direct likelihood comparison would lead to the same result. We have included it in STAR since distributions with more than 2 parameters will be implemented in a near future.

[12] To be honest we have to say that Fig. 3 is one of our rare examples where one of 6 duration distribution (the inverse Gaussian) is able to fit our data.

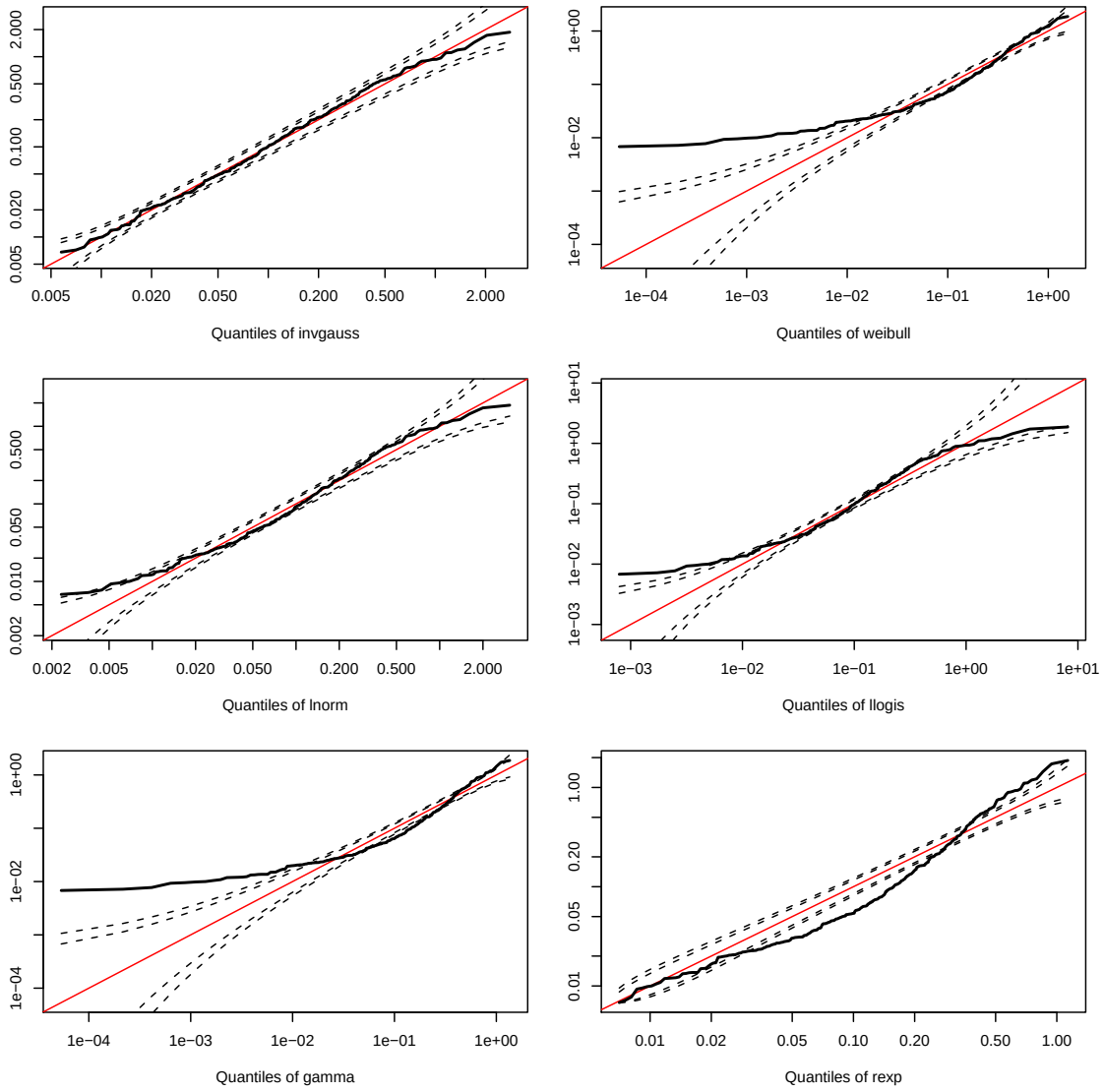

Figure 3: *theoretical quantile quantile plot*s of the six fitted duration distribution to the *isi*s of neuron 1, data set, e070528, spontaneous regime. The generation of this figure with STAR is explained in Sec. A.12.

2.4.5 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. 7 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

We next consider 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 formatted such that they are all locked with respect 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* [16]. 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 [25, 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 instantaneous firing rate*. For then if the successive responses of the neuron are uncorrelated and are "collapsed" or aggregated[13], the resulting process tends to an *inhomogenous Poisson process* [44, 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 ν is allowed to be time dependent. That is, Eq. 2 has to be modified as follows:

$$\text{Prob}\{N(t+\tau) - N(t) = n\} = \frac{\left(\int_t^{t+\tau} \nu(u)\, du\right)^n}{n!} \exp\left(-\int_t^{t+\tau} \nu(u)\, du\right) \tag{9}$$

where we see that if $\nu(u) = \text{Cst}$, Eq. 2 is obtained. The traditional *psth* produces a ν estimate that is a step function (ν is supposed to be constant over the time period spanned

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

by a bin). The *spsth* produces a smooth estimate of ν. 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 ν will be missed (filtered out) and the estimate will be biased, if set too small statistical fluctuations (due to the finite sample size) might be confounded with genuine changes of ν and the estimate will exhibit a large variance. 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. 16 illustrates this point[14].

The *smooth peri stimulus time histogram* of the present paper produces an estimate, $\hat{\nu}(t)$, of $\nu(t)$ through a compromise between two antagonistic goals: goodness of fit *and* smoothness (alternative, Bayesian, methods are available to built *spsth*s [25, 47]). If $\{x_i\}_{i=1}^N$ is the set of bin centers used in the preprocessing stage, δ, the bin width, M, the number of stimulus presentations and if $\{y_i\}_{i=1}^N$ is the set of corresponding spike counts, the *log likelihood* of $\hat{\nu}$ is:

$$\mathcal{L}(\{\hat{\nu}(x_i)\}_{i=1}^N) = \sum_{i=1}^{N} \Big(y_i \, \log\big(\hat{\nu}(x_i)\,\delta\,M\big) - \hat{\nu}(x_i)\,\delta\,M \Big) \tag{10}$$

Remember that the likelihood is *proportional* to the probability of the data. We are assuming here that the bin width, δ, is small enough to approximate the integral appearing in Eq. 9 with:

$$\int_{x_i-\frac{\delta}{2}}^{x_i+\frac{\delta}{2}} \nu(u)\,du \approx \nu(x_i)\,\delta \quad \forall\, i \in \{1,\ldots,N\}.$$

This *log likelihood function* (Eq. 10) is maximized by a step function whose step height in each bin is given by $\hat{\nu}(x_i) = \frac{y_i}{\delta\,M}$. That is what the classical *psth* does. But we want here a *smooth* estimate of ν. This is justified physiologically when the averaged response is estimated from neurons getting their input after a rather long sequence of events: odor molecule adsorption on the antenna's cuticle, transduction involving second messengers in the olfactory receptor neurons, spike propagation along the antenna's length, synaptic transmission from the receptors to the recorded antennal lobe neurons, feedback activity generated within the antennal lobe [23]. Each one of these stages would smear out in time its input even if the latter was a perfect step function. The smoothness hypothesis is therefore not a mere mathematically convenient requirement but a physiologically grounded assumption. We would moreover expect it to hold in a wide range of neurophysiological studies and definitely in any setting involving neurones recorded after the receptors stage. But arguing that smoothness makes sense does not give us directly a way of enforcing it as a constraint in our mathematical formulation. There is moreover another obvious constraint that ν, being an instantaneous firing rate, must satisfy: it must be non-negative. This latter constraint is dealt with as we did previously for the non-negative parameters of our duration distributions (Sec. 2.4.4): the log of ν is estimated directly. To make subsequent equations shorter we will write:

$$\eta() = \log\big(\nu()\,\delta\,M\big) \tag{11}$$

[14] But the reader should keep in mind that if the bin width is chosen *after* seeing the data, the interpretation of the pointwise 95% confidence interval as: "The true mean value of $\nu()$ over the bin should fall in the confidence interval 95 out of a 100 experimental replication", *is not valid anymore*.

Our previous log likelihood equation, Eq. 10 becomes:

$$\mathcal{L}(\{\hat{\eta}(x_i)\}_{i=1}^N) = \sum_{i=1}^{N} \Big(y_i\, \hat{\eta}(x_i) - \exp\big(\hat{\eta}(x_i)\big) \Big) \tag{12}$$

In order to benefit from the powerful *smoothing spline* results [46, 18] we are going to enforce our smoothness constraint by forcing the integral of the squared second derivative of $\hat{\eta}$ to be finite, that is:

$$\int \big(\frac{d^2\hat{\eta}(x)}{dx^2}\big)^2\, dx \;<\; \infty, \tag{13}$$

where the integral is evaluated on a domain containing all the x_is. We could here also use first order derivatives as well as third or higher order. The key requirement is to take the *integral of the square* of the chosen derivative. Choosing higher order derivatives leads to longer and more "memory hungry" computations as well as to estimates for which, by definition, derivatives at higher order do exist. Notice that the penalization is uniform on the definition domain of x. This is a potential weakness of the approach since for instance the *spsth* would ideally be flat prior to the stimulus onset which would lead one to want to constrain the pre-stimulus period to be smoother than the subsequent one. We combine our "goodness of fit requirement" (Eq. 12) with our "smoothness" requirement (Eq. 13) by finding the smooth function $\hat{\eta}$ *minimizing the penalized likelihood*:

$$-\sum_{i=1}^{N} \Big(y_i\, \hat{\eta}(x_i) - \exp\big(\hat{\eta}(x_i)\big) \Big) + \lambda \int \big(\frac{d^2\hat{\eta}(x)}{dx^2}\big)^2\, dx, \tag{14}$$

where the *smoothing parameter*, λ, is non negative. The first term here is the *opposite* of our log likelihood defined by Eq. 12. Minimizing this term amounts to maximizing the log likelihood and therefore to have a "good" fit. The second term is positive or null. It is null regardless of the value of λ when $\hat{\nu}$ is a linear function (whose graph is a straight line). As shown on Fig. 4 the chosen value of λ will have a crucial influence on the estimated function. A small value will lead to a function matching arbitrarily precisely the data while a large value will lead to a linear function (see also Fig. 4.1-4.3 [46, pp 46-47] and Fig. 1.1 [18, p 3]).

Three key results As mentioned before the *smoothing spline* theory is, thanks to the work of Grace Wahba and her collaborators [46, 18], extremely well developed. This is the availability of a strong theoretical results together with the availability of software which lead us to use this methodology. We will state here three results which are particularly important.

Solution of the penalized likelihood problem It can be shown [26], [46, Chap. 1], [18, Chap. 2 and 5] that the function $\hat{\eta}_\lambda$ minimizing Eq. 14 for a fixed λ exists in a *finite dimensional space* whose basis functions can be found. That is, $\hat{\eta}_\lambda$ can be written as:

$$\hat{\eta}_\lambda(x) = \sum_{j=0}^{1} d_j\, \phi_j(x) \;+\; \sum_{i=1}^{N} c_i\, R(x_i, x)\,, \tag{15}$$

where, assuming that a smoothness constraint on the second order derivative is used (Eq. 13) and that the x_is have been rescaled such that their definition domain is $[0, 1]$:

$$\phi_0(x) = 1 \quad \phi_1(x) = x - 0.5$$

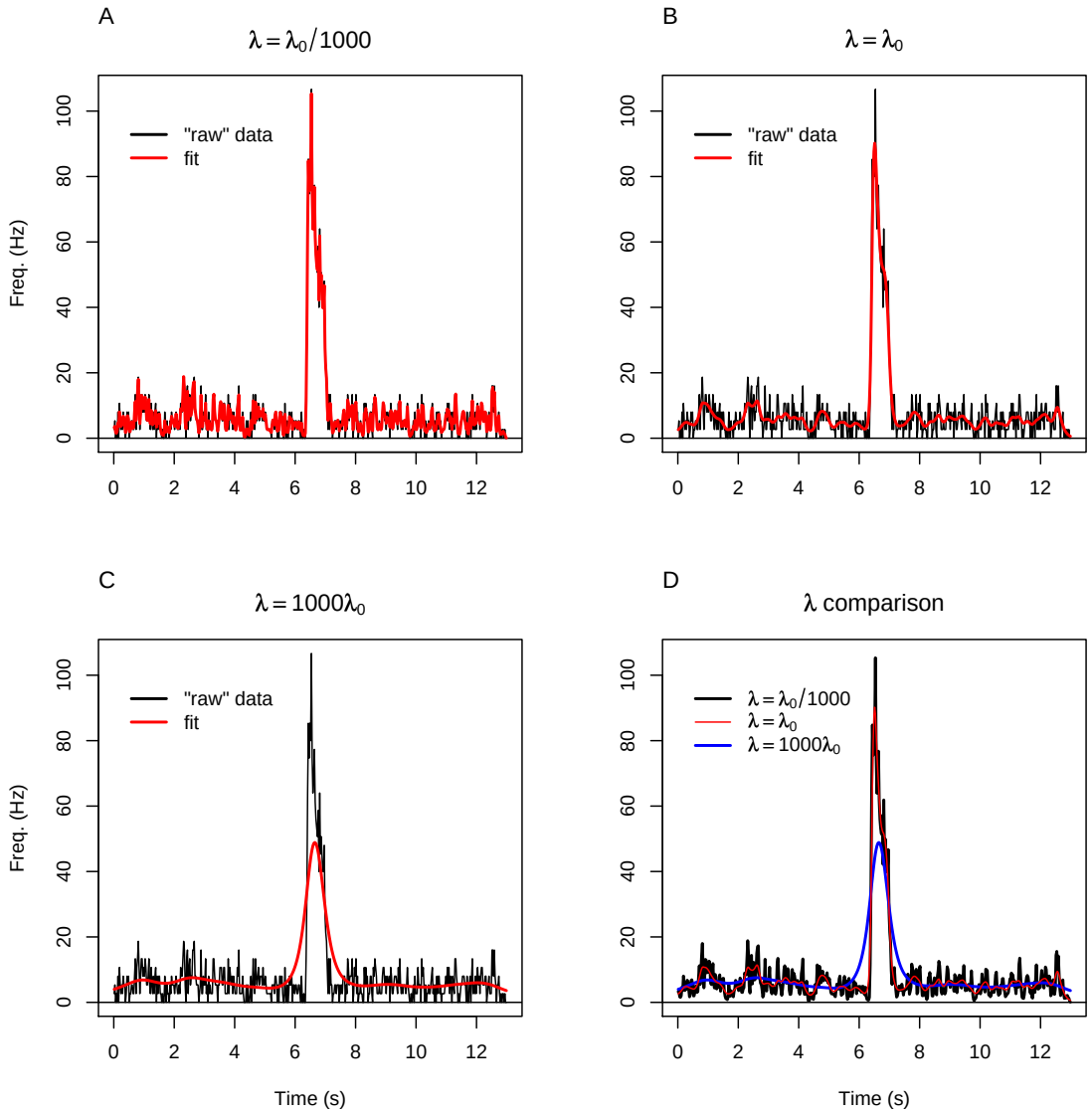

Figure 4: Influence of *smoothing parameter* choice. Comparison of fitted *smooth peri stimulus time histograms* using 3 different values for the *smoothing parameter* λ. An optimal value, λ_0 was obtained by the cross-validation method described in the sequel. Two additional were performed with $\frac{\lambda_0}{1000}$ (A) and with $1000\,\lambda_0$ (C). On A,B and C, the "raw data" (classical psth built with a bin width of 25 ms) appear as black and the fitted curve appears as red. D, the 3 fits superposed. A small λ leads clearly to a "wiggly" fit following the data closely, while a large one generates a fit following only the very slow variations of the discharge.

and

$$R(x_i,x) = [(x-\frac{1}{2})^2-\frac{1}{12}]\,[(x_i-\frac{1}{2})^2-\frac{1}{12}]/4-[(\mid x_i-x \mid -\frac{1}{2})^4-\frac{1}{2}(\mid x_i-x \mid -\frac{1}{2})^2+\frac{7}{240}]/24$$

The coefficients, d_0, d_1 and $\{c_i\}_{i=1}^N$ have to be found but classical methods for generalized linear models with penalization, *penalized iteratively re-weighted least squares* (PIRLS), can be directly used for that [18, Chap. 3 and 5] as detailed in Sec. B of the Appendix. The user of these methods should moreover be aware that at least 2 symmetric matrices whose approximate size are N^2 have to be allocated during the calculations. This is a strong incentive to keep our preprocessing bin size δ (Eq. 10) as "large" as possible. If we have an acquisition duration of 15 s for each stimulus presentation and if we use a δ of 25 ms we are going to need $2 \cdot (40 \cdot 15)^2 \approx 2.9$ MB of RAM. If we use instead a δ of 1 ms we are going to need 25^2 more, that is, roughly 1.8 GB. The first memory requirement is negligible for a modern PC, the second is likely to induce a "catastrophic" computation slowdown due to swap space filling if a "tiny" laptop is used.

Smoothing parameter selection by "performance-oriented iteration" The practical use of the *smoothing spline* methodology requires the availability of an efficient method for setting the *smoothing parameter*, λ (Fig. 4). Assuming that the data where generated by η, we define the "optimal" λ_0 as the one which minimizes:

$$\frac{1}{N}\sum_{i=1}^{N}\Big(\exp\big(\eta(x_i)\big)-\exp\big(\eta_\lambda(x_i)\big)\Big)\cdot\Big(\eta(x_i)-\eta_\lambda(x_i)\Big)$$

which is the average symmetrize Kullback-Leibler distance over the sample points [18, p 152]. Notice that we do not assume here that the "true" η function satisfies Eq. 13, it could for instance be discontinuous. The PIRLS algorithm used to estimate $\widehat{\eta}_\lambda$ at fixed λ can be combined with the *cross-validation* approaches used to choose λ in *penalized weighted least squares* (PWLS). Each iteration of the PIRLS algorithm is a Newton iteration where a new $\eta_\lambda^{(k+1)}$ is found as the minimizer of the quadratic approximation of the negative (penalized) log likelihood surface at the present $\eta_\lambda^{(k)}$ estimate. But the structure of the quadratic approximation is the same as the one of a PWLS. The idea of the *performance-oriented iteration* is then to solve a succession of PWLS problems by updating simultaneously the coefficients d_0, d_1, $\{c_i\}_{i=1}^N$ *and* the smoothing parameter. The cross-validation scheme used to estimate λ can be shown to give on average the optimal λ_0 in the PWLS setting [11, 46, 18]. In the present setting, strong arguments together with simulations suggests that the same holds but no formal proof is available [17]. Monte-Carlo simulations presented in Sec. 3.2.3, Fig. 10 A, demonstrate that the *performance-oriented iteration* is performing very well in the present setting.

For completeness we mention here that theoretically better cross-validation schemes have been designed for the present setting [48, 20]. These schemes do moreover outperform the present *performance-oriented iteration* scheme in multidimensional problems. They can moreover be used in our software by calling function `gsspsth` instead of `gsspsth0`. On our data we have nevertheless found that they do not improve the estimation of λ while they require more computation time.

Confidence intervals One of the most surprising results of the *smoothing spline* theory is that "confidence intervals" can be obtained [45, 34, 46, 18]. That is, a confidence band can be constructed around $\hat{\eta}_\lambda$ such that 95% of the $\{\eta(x_i)\}_{i=1}^N$ (where η is again the truth) are contained in it *on average*. The average coverage probability is exact in the Gaussian regression case and approximate in the present Poisson regression case. Monte-Carlo simulations presented in the Results section demonstrate that they are accurate on average

in the present setting. *The pointwise coverage probability is nevertheless not uniform.* A well known shortcoming for smoothing spline applied to Gaussian regression [45, 34, 12]. Using Monte Carlo simulations we show that the same goes in the present setting.

Numerical implementation All the numerical routines required to implement the *smoothing spline* methodology of this paper (and much more) are readily available in Chong Gu's user contributed R package: `gss` (for *general smoothing spline*) [18, 19]. The STAR function, `gsspsth0`, used to generate the *smooth peri stimulus time histograms* of this manuscript does a very simple job indeed:

1. Preprocessing: Given a bin size, δ, whose default is 25 ms, build an *unnormalized* classical *psth* , that is, create a vector, `Time`, containing the centers of the bins and a vector, `Count`, containing the total number of spikes (from all the trials) in each bin.

2. Call `gss` function `gssanova0` with `Count` as the dependent variable and `Time` as the independent one, specifying that the `Poisson` family has to be used.

3. Use `gss` function/method `predict.gssanova0` (see Sec. A.7 of the Appendix) on the output of `gssanov0` to get the estimate $\widehat{\eta}_\lambda$ and its associated standard error: $se_{\widehat{\eta}_\lambda}$.

4. Postprocessing: Get the estimated instantaneous frequency:

$$\widehat{\nu}_\lambda = \frac{\mathrm{e}^{\widehat{\eta}_\lambda}}{\delta\, M}$$

   and the upper / lower limit of the 95 % confidence band:

$$\frac{\mathrm{e}^{\widehat{\eta}_\lambda \pm 1.96\, se_{\widehat{\eta}_\lambda}}}{\delta\, M}$$

2.5.3 Empirical check of the *spsth smoothing parameter* and confidence intervals by Monte-Carlo simulations

In order to check the validity of the *smoothing parameter* selection and of the confidence intervals (or confidence bands) of our *spsth*s we used four data sets which are distributed with our package STAR (Sec. 2.1.4). For each neuron / data set combination (*e.g.*, `e070528citronellal_1`) the following simulations and analysis were carried out:

1. Get a *spsth*, $\hat{\nu}_\lambda$, using bin width of 25 ms which is the default value for this parameter in STAR (Fig. 5 A to B).

2. Extract the *smoothing parameter* value from the fit result. This value will be the target "best value" for the simulation: λ_0.

3. For $j = 1 : R$ (R equals 999 or 1000) do:

   (a) Simulate with the *thinning method* [13, pp 253-256], ideal spike trains whose discharge is an inhomogeneous Poisson process of instantaneous rate: $\hat{\nu}_\lambda$. The number of simulated trains is identical to the number of odor presentations used in the original data set (Fig. 5 B to C).

   (b) Fit a *spsth* to the simulated data using a bin width of 25 ms to get: $\lambda^{(j)}$ and $\tilde{\nu}_\lambda^{(j)}$ (Fig. 5 D).

   (c) Built the 95% confidence bands around $\tilde{\nu}_\lambda^{(j)}$ and count the number of points of $\{\hat{\nu}_\lambda(x_i)\}_{i=1}^N$ which are inside the band (Fig. 5 D).

4. Build a box plot or an histogram of the empirical λs as well as of the fraction of points of the true curve which are inside the confidence band.

5. Build the estimate of $\mathrm{E}(\tilde{\eta}_\lambda)$:

$$\tilde{\eta}_{\lambda,\bullet}(x_i) = \frac{1}{R}\sum_{j=1}^{R}\tilde{\eta}_\lambda^{(j)}(x_i) \quad i \in \{1,\ldots,N\}$$

   to obtain the pointwise *bias* estimate:

$$b(x_i) = \hat{\eta}_\lambda(x_i) - \tilde{\eta}_{\lambda,\bullet}(x_i) \tag{16}$$

6. Build an estimate of the pointwise coverage probability for a nominal 95 % confidence interval:

$$cp(x_i) = 1 - \frac{\#\{\hat{\nu}_\lambda(x_i) \notin CI(x_i)^{(j)}\}_{j=1}^{R}}{R} \tag{17}$$

   where $CI(x_i)^{(j)}$ stands for the 95 % confidence interval built around $\tilde{\nu}^{(j)}$ at point x_i.

These simulations where carried out on a multi CPU desktop computer (Intel Core 2 Quad, 2.4 GHz with 8 Go RAM). In order to exploit fully the computing power of the machine, 3 to 4 of the 4 CPUs were used simultaneously by parallelazing the above loop. That is when 3, respectively 4, CPUs were used a total of 999, respectively 1000, data sets were simulated by 3, respectively 4 subprocesses of R each one carrying out 333, respectively 250, simulations. This trivial parallelization is very easily implemented in R with the user contributed package, `snow` (for *simple network of workstation*), of Tierney, Rossini, Li and Sevcikova [42, 41]. An additional element is required to ensure good statistical properties of these parallel simulations: a random number generator able to generate statistically independent streams of random numbers. Pierre L'Écuyer and collaborators [29] developed such a generator which has been made available as an R user contributed package by Leydold and L'Écuyer [28, 30].

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 [33, Chap. 3]. 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, thanks to Eric Lecoutre's package R2HTML [27], one can write on HTML file without having to really learn HTML. Lecoutre short 2003 paper [27] 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.

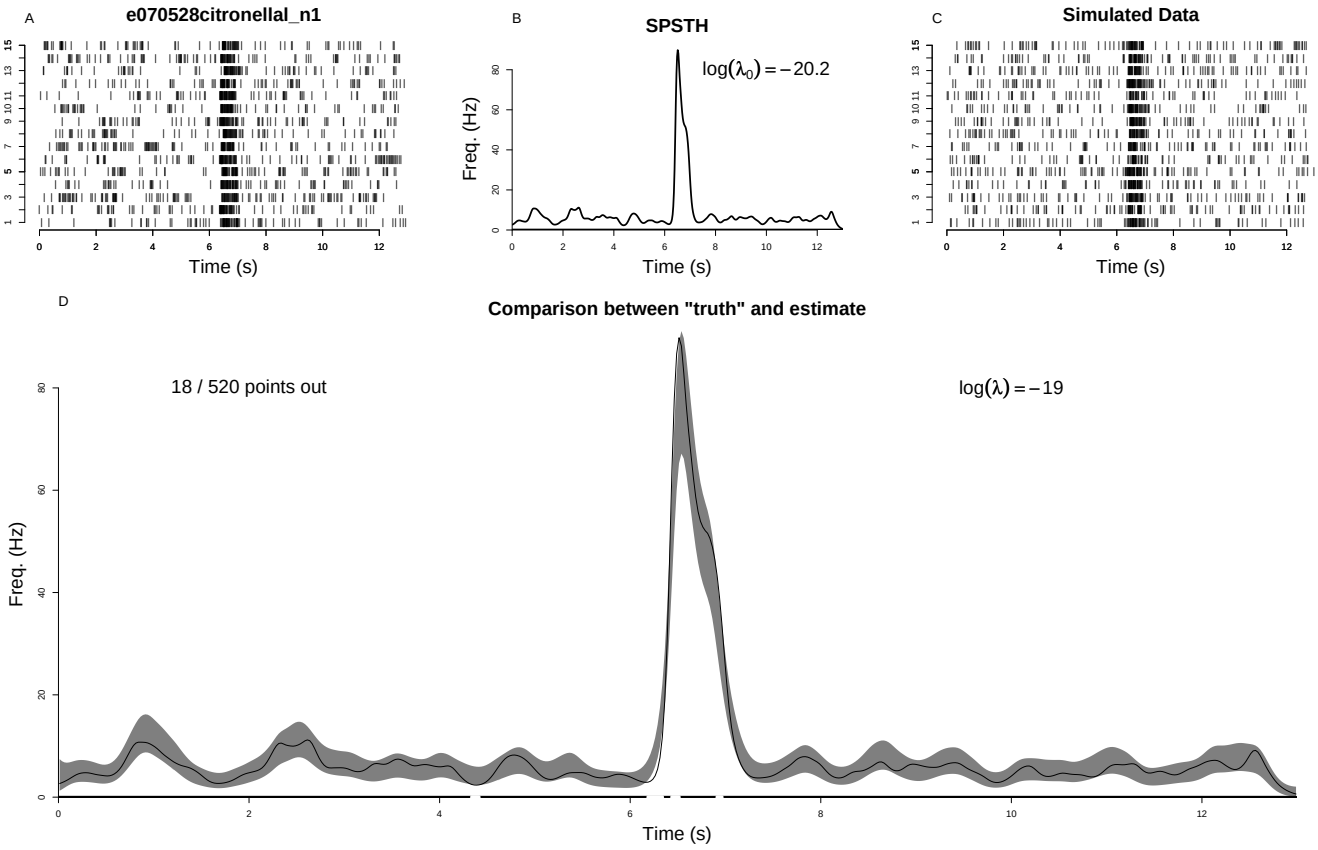

Figure 5: Simulation principle. A, raster plot of one of the neurons in our data set. B, the *smooth peri stimulus time histogram* together with the log of the estimated λ. C, One data set simulated as an inhomogeneous Poisson process using the *smooth peri stimulus time histogram* of B. D, grey band: the 95% confidence band obtained from the the fit of the *simulated data*; black curve: the "truth", that is the *smooth peri stimulus time histogram* used to simulate the data. The X axis has been blanked at the locations where the "truth" is out of the confidence band. The number of points of the truth which are out of the band is indicated together with the total number of points making the *smooth peri stimulus time histogram*. The log of the estimated λ is also shown. Notice two features on D: The estimated penalty weight, λ, is larger than the true one, λ_0, and the grey band is slightly smoother than the truth. The locations where the truth is out of the confidence bands seem to be preferentially associated with times at which the truth undergoes a "fast" slope change. This impression is correct as will be shown in Sec. 3.2.4.

2.7 Computers and software versions used for the analysis

Unless otherwise stated (*e.g.*, Sec. 2.5.3) the analysis described in this manuscript was carried out on laptop PC running Linux (Ubuntu 8.04.1). R version 2.7.2 was compiled with `gcc`[15] 4.3.1 using R's default compilation flags. The hardware characteristics of the laptop were: a dual core CPU (Intel Core 2) at 2 GHz with 1 GB of RAM.

2.8 Reproducing the present analysis

We are trying to implement the rather strict "reproducible research" paradigm of [15, Gentleman and Temple-Lang, 2004]. This means that in addition to giving access to the software (STAR) and to the data sets used in the present paper we provide a `compendium` [15] which is fundamentally a metafile containing both the text of the present paper and the R instructions required to *regenerate* all the figures and tables of the paper. By editing this metafile the interested reader can have access to the complete description of the computations involved in the paper and change them if he/she wants. Like the data sets, the metafile is distributed with the package. The user contributed package `cacheSweave` of Roger Peng [36] has been a tremendous help for developing our compendium.

3 Results

The general features of our data analysis procedures are illustrated and characterized by applying them to each data set of Sec. 2.1.4. The automatic report generation is illustrated using 2 data sets: e070528spont_4 for the "spontaneous regime" data and e070528citronellal_1 for the "stimulus response regime" data.

3.1 Spontaneous activity analysis

Run-time as well as summary data obtained by running the same analysis on our 12 spontaneous data sets are presented first. The HTML report generated out of one of the data set is illustrated next.

3.1.1 Robustness and run-time of spontaneous activity analysis

We do not have a formal proof of the "robustness" of our analysis routines for the gamma, Weibull and loglogistic models but the systematic application of the same automatic methods to diverse data sets can be used to argue in favor of it. The renewal test plots (Sec. 2.4.3) and the 6 bivariate duration distribution models (Sec. 2.4.4) were therefore systematically generated and fitted to the 12 spontaneous spike trains described in Sec. 2.1.4. In all the cases the analysis did run smoothly. This data set contains moreover only one spike train for which one of the bivariate duration density models can be considered as reasonable as judged by the Q-Q plot (Sec. 2.4.4, Fig. 3). Table 1 contains the run-time results. The run-time of report generation is also given for each data set. The latter time is much larger than the previous ones because the Q-Q plots are included in the report as well as the smooth version of the cross-correlograms. As expected the computation time is proportional to the number of spikes in the train with an increase of 0.52, 0.24 and of 38 ms per spike for the renewal test plot, the fits and the reports respectively.

3.1.2 Example of automatic report generation

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

[15] http://gcc.gnu.org

	N	Renewal	Fit	Report
e060517spont_1	356	0.54	0.22	11.24
e060517spont_2	490	0.31	0.17	12.44
e060517spont_3	216	0.15	0.08	6.50
e060817spont_1	529	0.32	0.13	27.44
e060817spont_2	1229	0.71	0.37	47.49
e060817spont_3	781	0.46	0.23	28.82
e060824spont_1	505	0.31	0.14	10.21
e060824spont_2	64	0.08	0.05	3.25
e070528spont_1	336	0.22	0.09	19.33
e070528spont_2	1173	0.68	0.27	50.52
e070528spont_3	1834	1.06	0.50	65.29
e070528spont_4	1015	0.62	0.23	42.46

Table 1: Summary statistics of spontaneous spike train analysis. N: The number of spikes. Renewal: Time (s) required to carry out the computation and to generate the renewal test plot (Sec. 2.4.3). Fit: Time (s) required to fit the 6 duration density models (Sec. 2.4.4). Report: Time (s) required to generate an HTML report as detailed in Sec. 3.1.2

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[16] 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).

If other spike trains (from simultaneously recorded neurons) are provided, then *cross-correlation histograms* are estimated. Two estimations methods are available (Sec. 2.4.5), the classical histogram and a smooth version of it. Argument chh controls if a single estimation is performed or if both are performed. Fig. 7 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 roughly 100 ms before a spike in neuron 4 seems to be significant.

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

Most of the analysis presented in this section will consist in a systematic application of the same routines to each of the odor response data sets of STAR (Sec. 2.1.4). Fig. 8 shows the *smooth peri stimulus time histogram* obtained for each neuron in each data set using our default preprocessing bin width. On each plot the 0.5 s opening odor delivery valve comes around second 5. The reader can see that the data sets exhibit diverse type of responses (but more excitation than inhibition). The basal firing rates range from 5 to 40 Hz. The largest response is above 80 Hz.

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

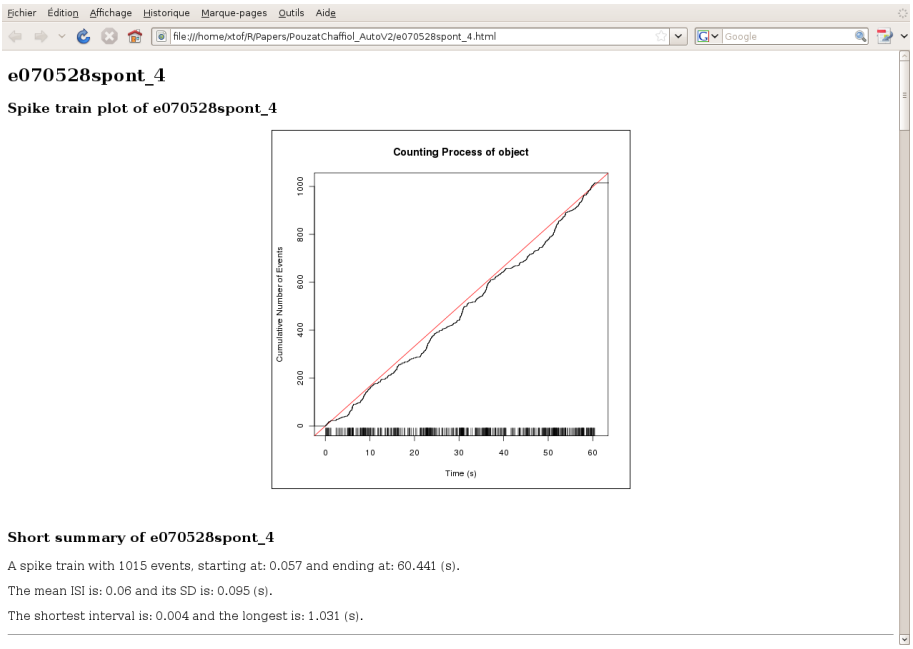

Figure 6: Fig. 1 as it appears in the automatically generated HTML report.

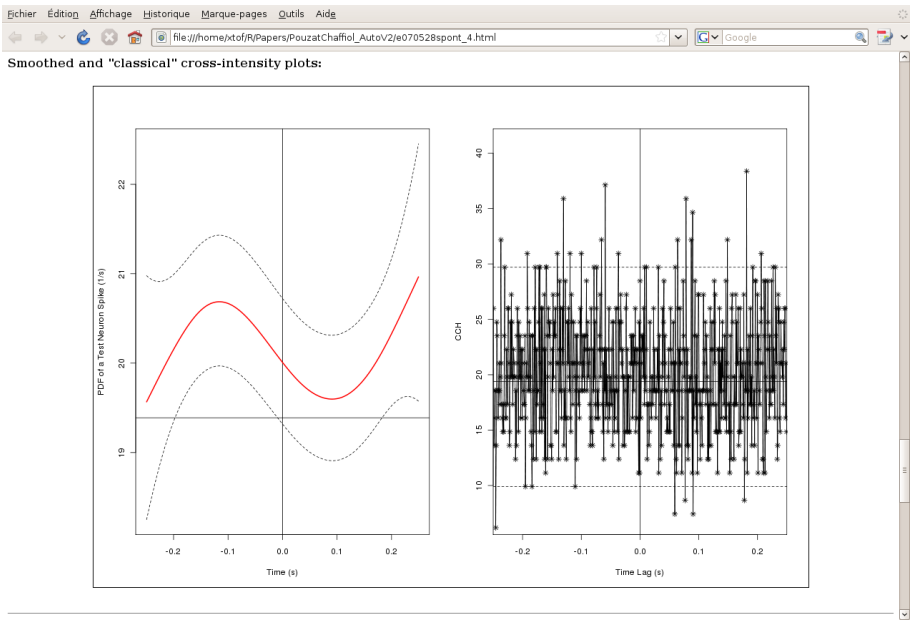

Figure 7: *Smooth* and "*classical*" *cross-correlation histograms* between neuron 4 (reference) and neuron 2 (test) as they appear in the automatically generated HTML report. One both plots the dotted lines define a pointwise 95 % confidence region.

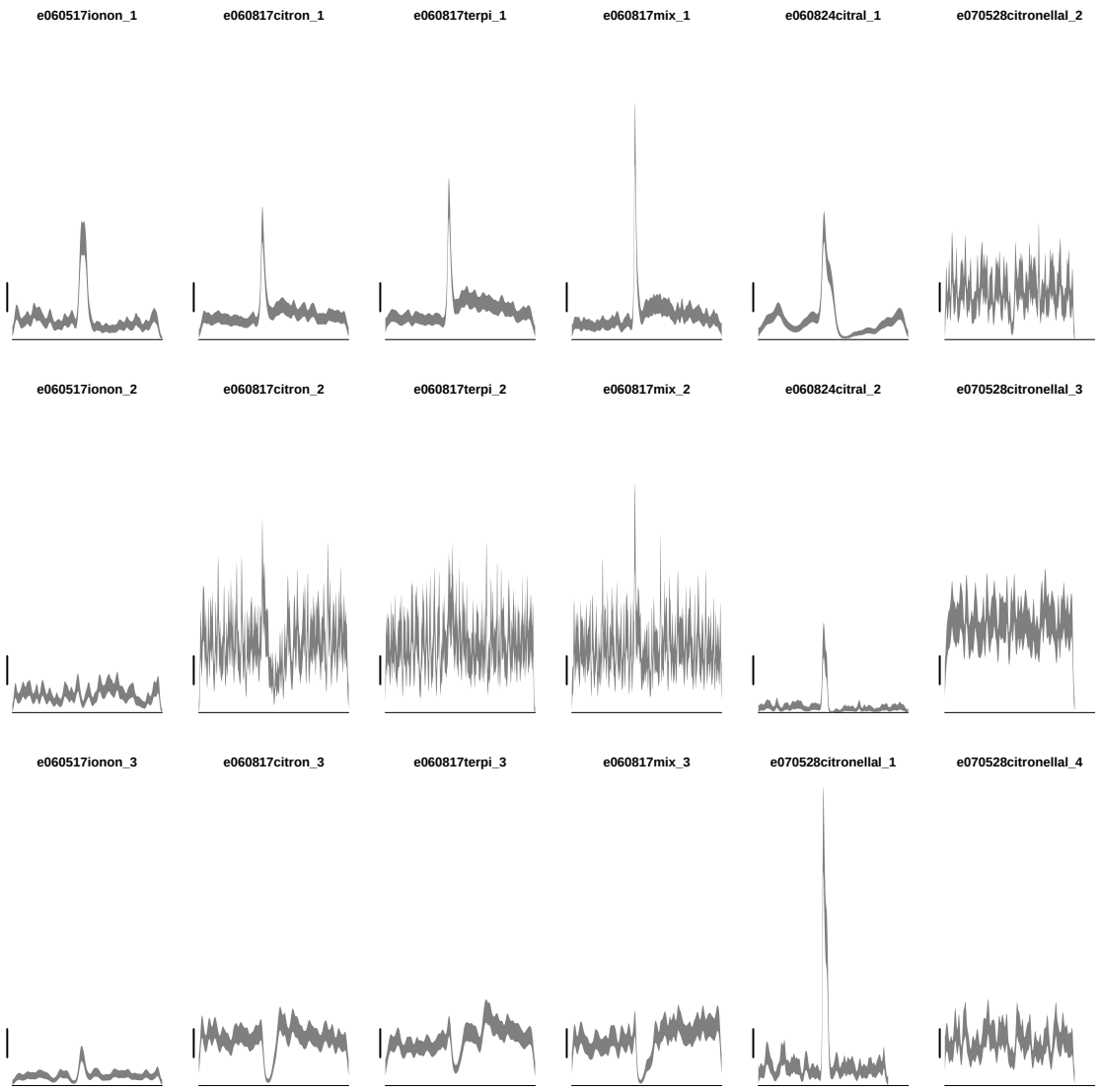

Figure 8: The 18 *smooth peri stimulus time histogram*s. All graphs have the same scale. The scale bar on the left side of each plot is drawn between 10 and 20 Hz. The time axis spans 15 s. The estimated *smooth peri stimulus time histogram*s are shown as a 95 % confidence band.

3.2.1 Robustness and run-time of stimulus response analysis

The approach followed in Sec. 3.1.1 to study the robustness of our routines for the analysis of the spontaneous regime is repeated here with the stimulus response regime. *Smooth peri stimulus time histograms* and HTML reports were generated using the default preprocessing binwidth (25 ms). In all the cases the analysis did run smoothly. Table 2 contains the run-time results. The computation time is proportional to the number of spikes in the train with an increase of 0.6 and of 0.74 ms per spike for the *smooth peri stimulus time histograms* and the reports respectively.

	N	S	Duration	SPSTH	Report
e060517ionon_1	2073	19	15.00	7.52	10.05
e060517ionon_2	1696	19	15.00	7.01	9.17
e060517ionon_3	977	19	15.00	6.23	8.75
e060817citron_1	2639	20	15.00	7.40	9.66
e060817citron_2	6920	20	15.00	10.65	13.92
e060817citron_3	4805	20	15.00	7.40	10.60
e060817terpi_1	3117	20	15.00	7.58	10.32
e060817terpi_2	6903	20	15.00	11.05	14.08
e060817terpi_3	4762	20	15.00	7.90	10.48
e060817mix_1	2515	20	15.00	7.12	10.16
e060817mix_2	6512	20	15.00	11.01	14.32
e060817mix_3	4771	20	15.00	7.92	10.81
e060824citral_1	2065	20	15.00	8.04	10.54
e060824citral_2	599	20	15.00	7.74	9.70
e070528citronellal_1	1596	15	13.00	5.36	7.89
e070528citronellal_2	3073	15	13.00	5.96	8.16
e070528citronellal_3	5884	15	13.00	7.20	9.44
e070528citronellal_4	2873	15	13.00	5.42	7.39

Table 2: Summary statistics of stimulus response analysis. N: The total number of spikes. S: The number of stimulations. Duration: Duration (s) of the acquisition for each stimulation. SPSTH: Time (s) required to compute the *smooth peri stimulus time histogram*. Report: Time (s) required to generate an HTML report.

3.2.2 Insensitivity to the preprocessing bin width

In order to put on a firmer basis our claim of insensitivity's of our *spsth* to the preprocessing bin width used, we show on Fig 9 the effect of a 5 times bin width reduction. We use the 3 neurons of data set `e060817mix` because a the variety of neuronal responses present in this data set (see also Fig. 8 column 4). In order to make the comparison between the two *spsth*s obtained with the two bin width clearer, we only show part of the recording window (from 5 to 10 s) and the "coarser" *spsth* generated with the largest bin width (25 ms, default value) appears as a 95 % confidence band. The "finer" *spsth* obtained with the smallest bin width (5 ms) is shown as a black curve. It can be seen than the finer *spsth* is "wigglier" and *within* the confidence band of the coarser. Using a 5 ms bin width instead of a 25 ms one will therefore not essentially change our estimation.

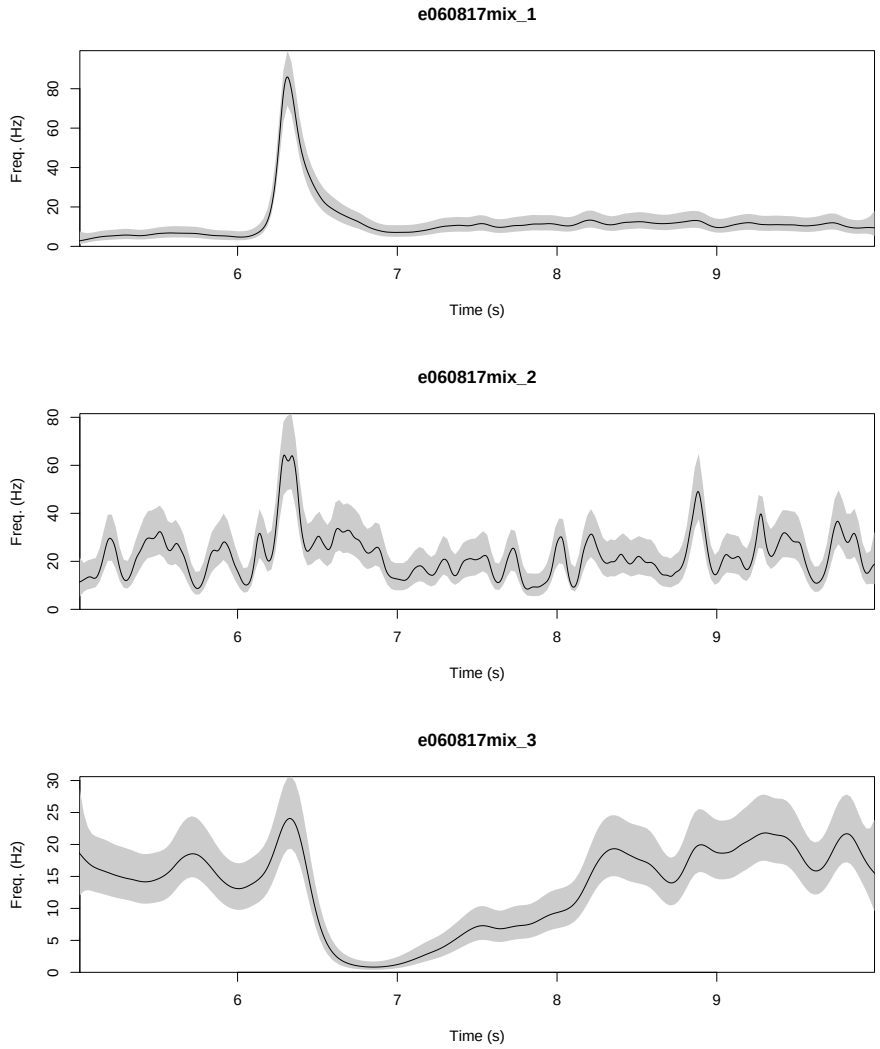

Figure 9: Absence of effect of the preprocessing bin width on the *spsth*s. The 3 neurons of data set `e060817mix` are illustrated (3rd column of Fig. 8). Each graph shows the *spsth* obtained with the default bin width (25 ms) as a grey confidence band. The *spsth* obtained with a preprocessing bin width of 5 ms is shown a black curve.

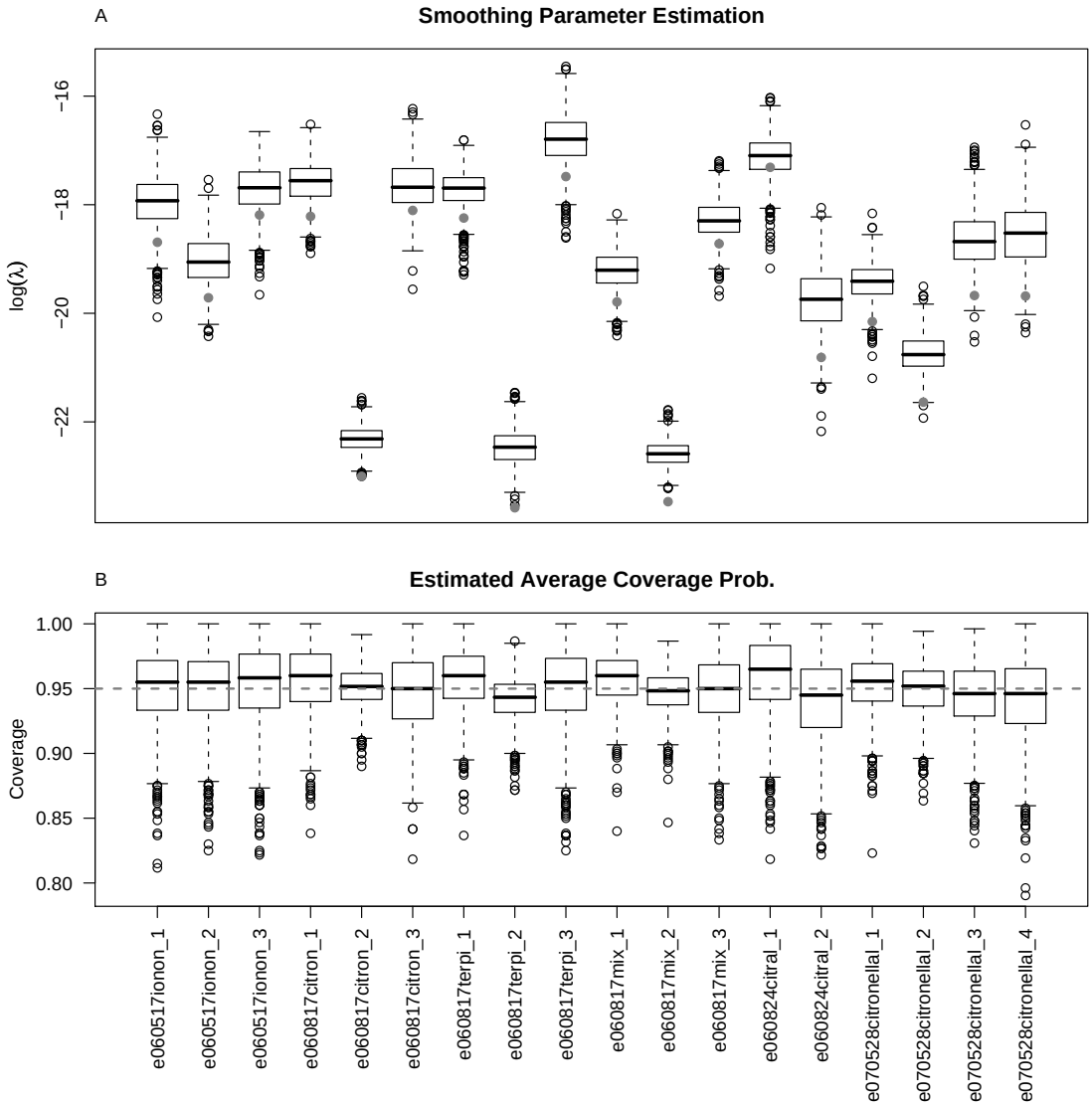

Figure 10: A, Estimated (box plot) and actual (grey point) value of $\log \lambda$. B, Estimated average coverage probability (box plot). The nominal value appears as a dotted line.

3.2.3 Monte Carlo investigation of *smoothing spline smoothing parameter* selection and average coverage probability for 95% confidence intervals

As described in Sec. 2.5.3 and Fig. 5 an empirical investigation of the cross-validation based *smoothing parameter* selection was carried out together with an empirical determination of the *average* coverage probability of the 95 % confidence intervals. The results of these Monte Carlo simulations are summarized on Fig. 10. It can be seen that the performance-oriented iteration algorithm slightly over smooths the data ($\lambda > \lambda_0$ in general). This could very well be due to the fact that our preprocessing stage (building a classical *psth* with a small bin width) does itself slightly smooth the data. This slight over smoothing does nevertheless not seem to have any consequence on the average coverage probability. But a relatively wide distribution of the average coverage probability is seen ranging from 100 to 80 %.

3.2.4 Monte Carlo investigation of the pointwise coverage probability of the 95% confidence intervals

Grace Wahba who introduced the method to compute the confidence intervals [45] also took pain insisting that these intervals must be interpreted "across the function" [46, Wahba, 1990, p 69], that is, they are valid on *average* over the x_is as shown on Fig. 10. In order to avoid wrong interpretations of the confidence intervals we show in this section the empirical *pointwise* coverage probability of the 95% confidence intervals in three cases. The method used to obtain these pointwise coverage probability is explained in Sec. 2.5.3. For each of our 18 cases we estimated the pointwise coverage probability and, in each case, we look for its smallest value (over the x_i). We then selected the three cases which gave us the three lowest values. These 3 cases are shown on Fig. 11, the worst case making the left column. We shown on this figure (top row) the "true" $\eta()$ instead of the $\nu()$ because that is what the algorithm is working with and actually estimating. The second row shows the estimated bias (Sec. 2.5.3), that is, the difference between the "truth" and the mean estimated function generated by our procedure. The bias is seen to be large where *the slope of* $\eta()$ *changes fast*. This is expected since we are penalizing large values of the second derivative of $\eta()$ (Eq. 13 and 14). Where the bias is "large", the pointwise coverage value (Fig. 11, bottom row) can be much smaller than the nominal value (0.95).

3.2.5 Example of automatic report generation

A single screen shot of the full report is shown here. The full report can be seen in our web site[17] or, even better, can be generated by the reader as explained in Sec. A.18. 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 *spsth* (Sec. 2.5.2) is superposed to it (see Sec. A.17 for details). Fig. 12 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 pre-processing was given the estimated rate. 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. 17). A Graph showing the results of the Ogata's battery of tests for point processes [35] when the *spsth* is taken has a model of the neuron's response to *individual* stimulation is added next (not shown). We do not discuss further this last point here given that showing that actual *non averaged* neuronal discharge are not well described by *inhomogenous Poisson process* is somewhat showing the obvious. This tests will be detailed elsewhere. The impatient reader can of course already check the help files of STAR.

4 Discussion

Experimental techniques used in modern neuroscience 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

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

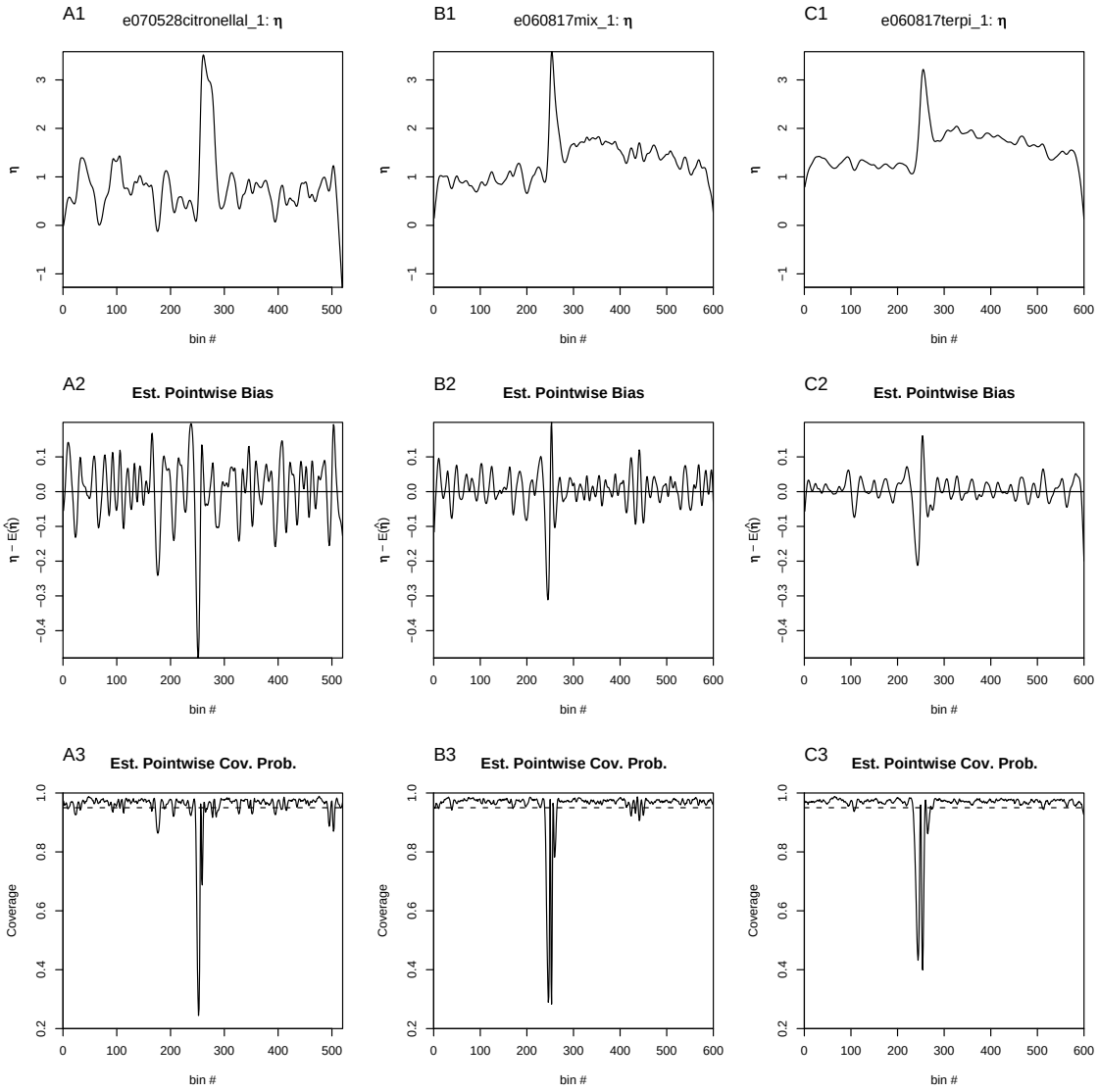

Figure 11: The three cases giving the three lowest minimal $cp(x_i)$. Top row (A1, B1,C1): The "true" η, that is, the one used to simulate the data. Central row (A2, B2,C2): The estimated pointwise bias, $b(x_i)$ (Eq. 16). Bottom row (A3, B3, C3): The estimated pointwise coverage probabilities for a 95 % confidence interval. Dotted line: the nominal value. On a given row, all graphs have the same scale.

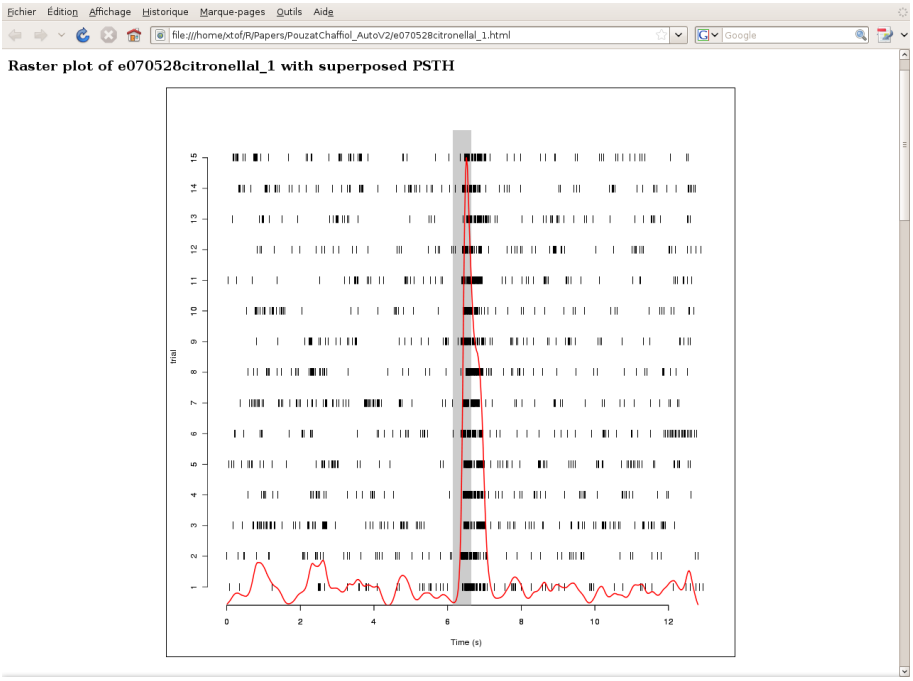

Figure 12: Raster plot with superposed *smooth peri stimulus time histogram* (red curve) for neuron 1 as it appears in the automatically generated HTML report.

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 (Table 1 and 2). But we insist again on the preliminary aspect of this automatic analysis. Most of the data we showed require more sophisticated analysis techniques.

The use of the *smoothing spline* methodology has been decisive in the development of a robust automatic analysis. We have tried to justify this methodological choice on both "theoretical" (Sec. 2.5.2) and "practical" (Sec. 3.2.3 and 3.2.4) grounds. We have moreover tried to carefully illustrate the meaning of the confidence intervals obtained with this method (Sec. 3.2.4). Given these results we would recommend to draw *spsth*s as confidence bands in order to convey a better impression of what has been learned from the data. While looking at such bands, the reader should keep in mind that a correct pointwise confidence interval should be larger where the estimate undergoes a fast slope change. If a data analyst wanted to make a strong statement about differences of sharp peak on *spsth*s, we would recommend for now a Monte Carlo simulation to be carried out in order to correct for the bias of the estimated *spsth*. We could also consider implementing the method described by [12, Cummins et al, 2001] to set locally the *smoothing parameter* (and

get a uniform coverage probability). We have exposed here only "the tip of the iceberg" of what can be done with *smoothing spline* and spike trains. The methodology can be readily extended to multidimensional cases and provide an elegant and practical solution to, at least some, conditional intensity estimation problems (Pouzat, Chaffiol and Gu *in preparation*).

Acknowledgments

We thank Chong Gu, Laurent Moreaux, Romain Franconville and Alain Marty for comments on the manuscript. C. Pouzat was partly supported by a grant from the AFM/Decrypton project. A. Chaffiol was supported by a DGA (?) fellowship.

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 downloaded 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[18]. 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()
```

[18] 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:xtable"
 [3] "package:STAR"      "package:gss"
 [5] "package:R2HTML"    "package:mgcv"
 [7] "package:survival"  "package:splines"
 [9] "package:tools"     "package:stats"
[11] "package:graphics"  "package:grDevices"
[13] "package:utils"     "package:datasets"
[15] "package:methods"   "Autoloads"
[17] "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 explicitly 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 explicitly 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 language [21] *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 obscure 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[19]:

```
> 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.

[19] Remember that when R is 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 interpret 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[20]. 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 executed 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 extremely efficient concept. With it "casual" users can plot spikeTrain objects and obtain immediately 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 significant 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:

[20] 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 example 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 usefulness 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 Why is the rank more informative

In Sec. 2.4.3 we mention that plotting O_{j+1} as a function of O_j is better than plotting i_{j+1} as a function of i_j. Let us illustrate this point by generating the equivalent of the upper left plot of Fig. 2 in term of i_j instead of O_j to get Fig. 13:

```
> n4.isi <- diff(e070528spont[["neuron 4"]])
> plot(n4.isi[-length(n4.isi)], n4.isi[-1], pch = ".",
+     xlab = expression(i[j]), ylab = expression(i[j +
+         1]))
```

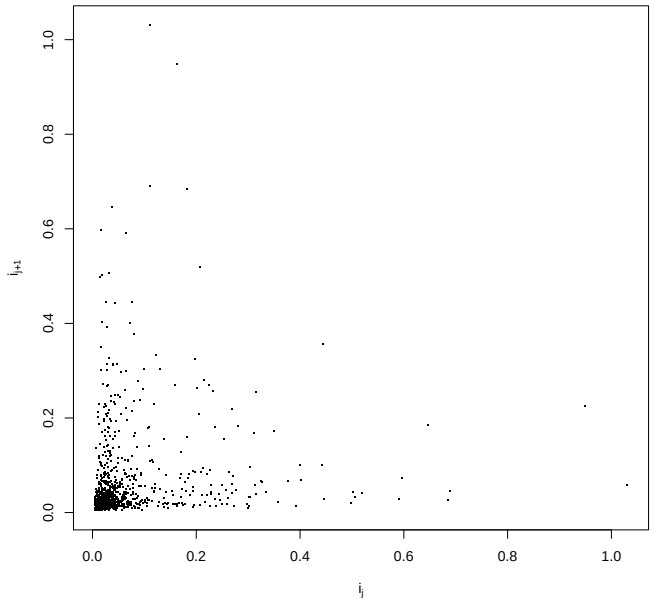

Figure 13: Plot of i_{j+1} versus i_j (in s) for neuron 4, data set, e070528, spontaneous regime. Compare with upper left plot of Fig. 2.

A.12 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
```

A.13 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. 14, we would enter:

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

A.14 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),
+     ...)
```

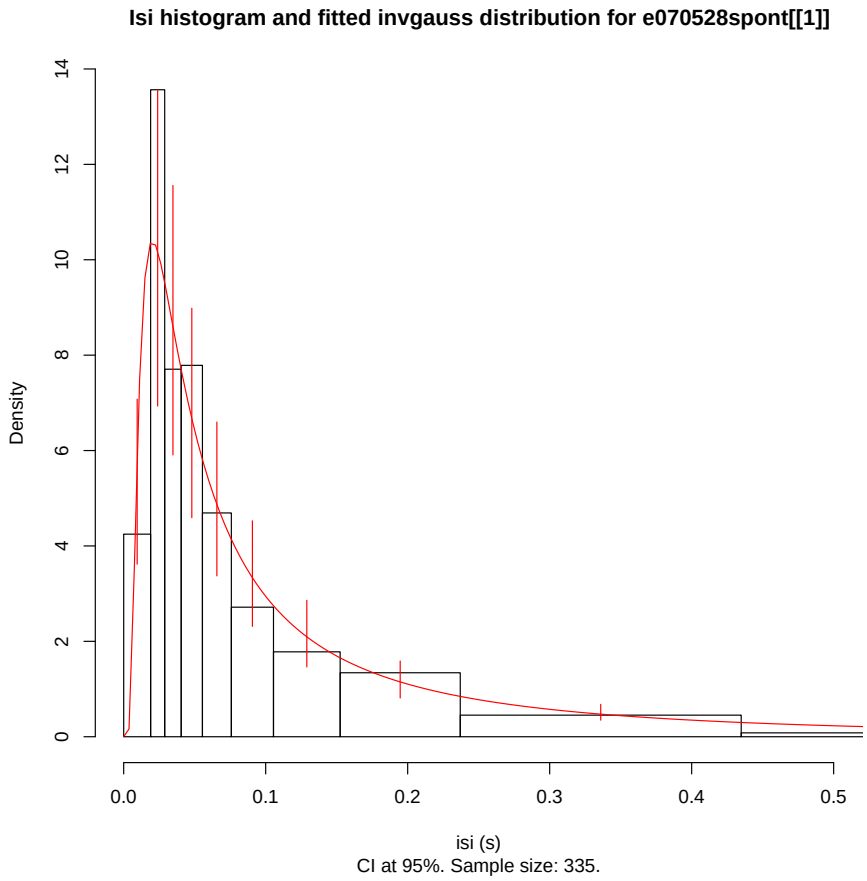

Figure 14: *isi* histogram (black rectangles) with superposed fitted inverse Gaussian density (red curve). The 10 histogram bins are set after the fit such that if the fit was good, 10% of the *isi*s would fall into each bin. Data of neuron 1, data set, e070528, spontaneous regime. For more details: ?isiHistFit.

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

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

Simple isn't it?

A.15 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. 15 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. 15 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[21] when matching functions arguments.

A.16 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. 16) 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))
```

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

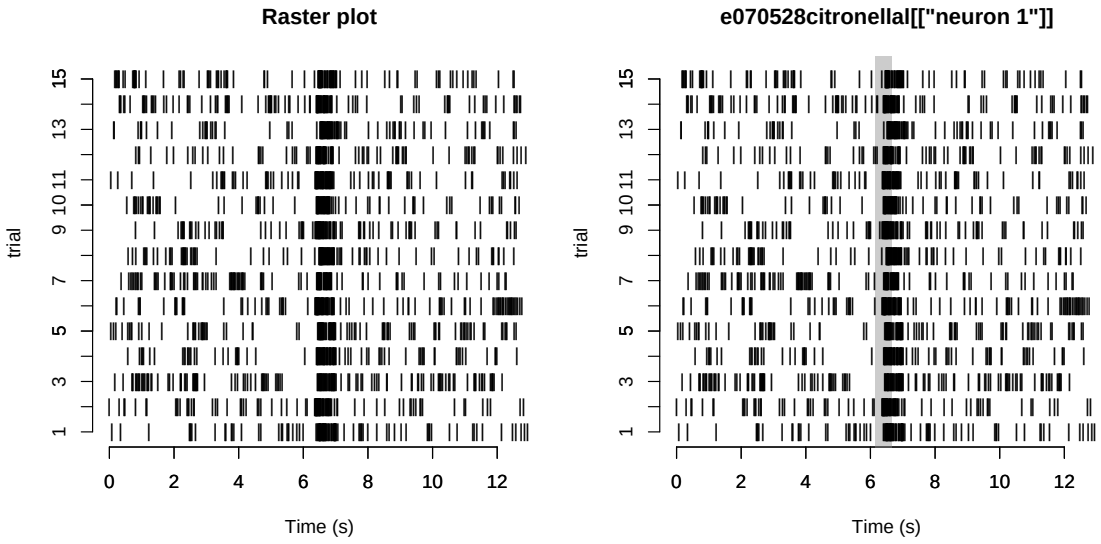

Figure 15: Illustration of print and plot methods for repeatedTrain objects. The 15 responses of neuron 1, data set e070528citronellal are used here. Left, plot generated by: e070528citronellal[["neuron 1"]]. Right, plot generated by: plot(e070528citronellal[["neuron 1"]],stim=c(6.14,6.64),main="e070528citronellal[[neuron 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).

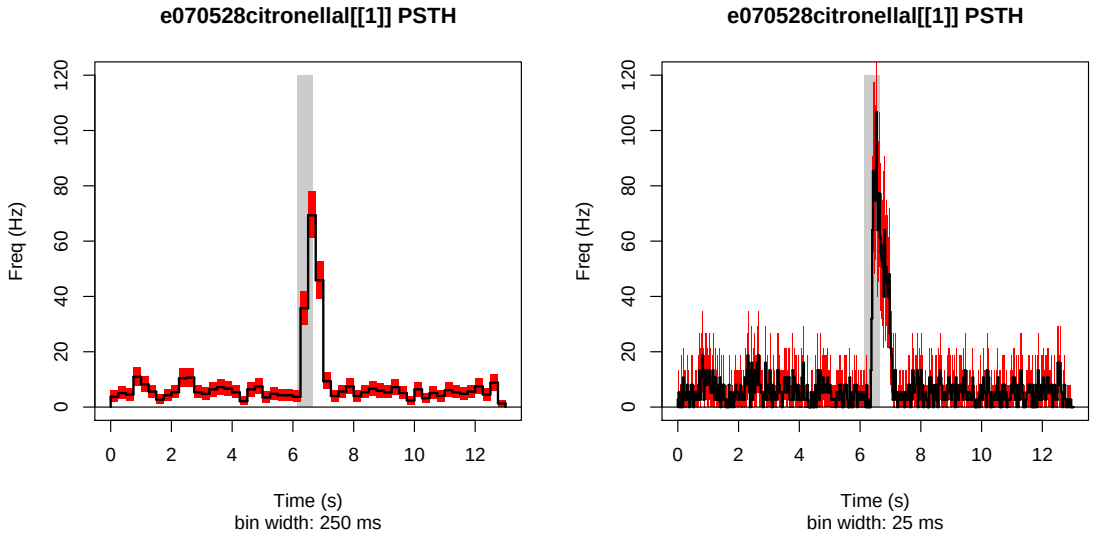

Figure 16: Illustration of psth function. The 15 responses of neuron 1, data set e070528citronellal are used here. Left, *psth* obtained with a bin width of 250 ms. Right, *psth* obtained with a bin width of 25 ms. The 95% confidence region appears in red.

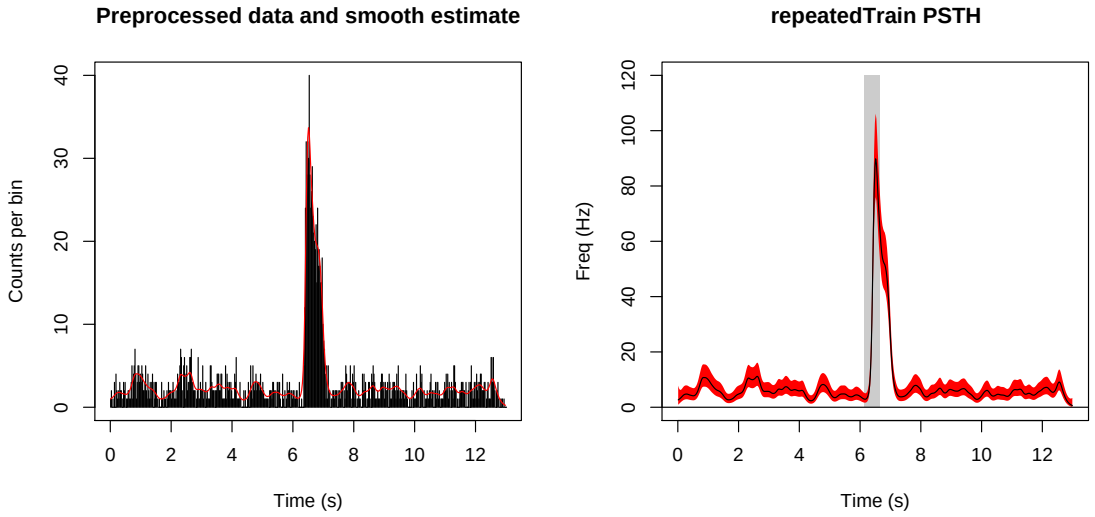

Figure 17: Illustration of gsspsth0 function. Same data as Fig. 16. Left, preprocessed data (black) used to obtain the "smooth" (red). Right, *spsth* obtained with function gsspsth0. The 95% confidence region appears in red. The Y axis scale has been adjusted to facilitate comparison with Fig. 16.

A.17 Details on *spsth*s

Smooth peri stimulus time histograms are obtained in STAR with the gsspsth0 function (see ?gsspsth0). 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 gsspsth0 calls function gss of Chong Gu's package gss after this preprocessing. gss does the real work of getting the *spsth*. It uses *smoothing spline* to get the smooth estimate.

Continuing with our previous example we would get a gsspsth0 object with:

```
> n1CitrGSSPSTH0 <- gsspsth0(e070528citronellal[[1]])
```

It can be worth comparing at that stage the preprocessed data out of which the "smooth" was constructed with the smooth itself. This is illustrated on Fig. 17 (Left) and obtained as follows:

```
> X <- n1CitrGSSPSTH0$mids
> Counts <- n1CitrGSSPSTH0$counts
> theBS <- diff(X)[1]
> nbTrials <- n1CitrGSSPSTH0$nbTrials
> Y <- n1CitrGSSPSTH0$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. 17, Right) is obtained with the plot method (plot.gsspsth0):

```
> plot(n1CitrGSSPSTH0, colCI = 2, lwd = 1, stim = c(6.14,
+     6.64), ylim = c(0, 120))
```

A.18 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 that we have created a subdirectory `report` in our current working directory and that we want to save our report there:

```
> reportHTML(object = e070528spont[["neuron 4"]],
+     directory = "report", filename = "e070528spont_4",
+     otherST = e070528spont[-4], laglim = c(-1,
+         1) * 0.25, forceTT = FALSE)

    gss warning in gssanova0: performance-oriented iteration fails to converge
```

Argument forceTT controls the generation of the *theoretical quantile quantile plot*s 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 histogram*s (by default). The lag of these *cross-correlation histogram*s is controlled by argument laglim. For more details, see ?reportHTML.spikeTrain. We see moreover see here that a warning is returned: `gss warning in gssanova0: performance-oriented iteration fails to converge`. After checking the documentations of reportHTML.spikeTrain, gsslockedTrain0 and gssanova0, we see that the argument `maxiter` of the latter has to be modified in order to perform enough iterations for convegence to be attained. This precisely what the "..." argument of reportHTML.spikeTrain allows us to do. We therefore repeat the report generation with an additional argument, `maxiter=60`, instead of the implicit default, `maxiter=30`, and see if the warning disappears:

```
> reportHTML(object = e070528spont[["neuron 4"]],
+     directory = "report", filename = "e070528spont_4",
+     otherST = e070528spont[-4], laglim = c(-1,
+         1) * 0.25, forceTT = FALSE, maxiter = 60)
```

We see that with, `maxiter=60`, enough iterations have been performed.

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

```
> reportHTML(object = e070528citronellal[["neuron 1"]],
+     directory = "report", filename = "e070528citronellal_1",
+     stim = c(6.14, 6.64))
```

A.19 Software versions used for this tutorial

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

```
R version 2.7.2 (2008-08-25)
i686-pc-linux-gnu

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

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

```
other attached packages:
[1] xtable_1.5-3    STAR_0.1-9      gss_1.0-1
[4] R2HTML_1.59     mgcv_1.4-1      survival_2.34-1
```

B Reproducing Fig. 4

Fig. 4 is obtained using the version of *penalized iteratively re-weighted least squares* described by [18, Gu, pp 62, 76 and 151]. We first define a function performing solving a penalized weighted least squares problem (together with some associated methods):

```
> ss_lambdaFixed <- function(y, x, lambda, w) {
+     rk <- function(x, y) {
+         k2 <- function(x) ((x - 0.5)^2 - 1/12)/2
+         k4 <- function(x) ((x - 0.5)^4 - (x -
+             0.5)^2/2 + 7/240)/24
+         k2(x) * k2(y) - k4(abs(x - y))
+     }
+     S <- cbind(rep(1, length(x)), x - 0.5)
+     Q <- outer(x, x, rk)
+     if (!missing(w)) {
+         S <- diag(sqrt(w)) %*% S
+         y <- sqrt(w) * y
+         Q <- diag(sqrt(w)) %*% Q %*% diag(sqrt(w))
+     }
+     else {
+         w <- NULL
+     }
+     Sqr <- qr(S)
+     R <- qr.R(Sqr)
+     theQ <- qr.Q(Sqr, complete = TRUE)
+     tF1 <- t(theQ[, 1:2])
+     F2 <- theQ[, -(1:2)]
+     rm(theQ, Sqr)
+     n <- length(y)
+     G <- chol(t(F2) %*% Q %*% F2 + n * lambda *
+         diag(n - 2))
+     u <- forwardsolve(t(G), t(F2) %*% y)
+     v <- backsolve(G, u)
+     c <- F2 %*% v
+     rightMember <- (tF1 %*% y) - (tF1 %*% Q %*%
+         c)
+     d <- backsolve(R, rightMember)
+     result <- list(x = x, y = y, lambda = lambda,
+         w = w, n = n, d = d, c = c)
+     environment(rk) <- globalenv()
+     result$rk <- rk
+     class(result) <- "simpleSmooth"
+     result
+ }
> plot.simpleSmooth <- function(x, y, ...) {
+     plot(x$x, x$y, ...)
```

```
+ }
> predict.simpleSmooth <- function(object, newx,
+     ...) {
+     if (missing(newx))
+         newx <- object$x
+     phi <- cbind(rep(1, length(newx)), newx -
+         0.5)
+     R <- outer(newx, object$x, object$rk)
+     if (!is.null(object$w))
+         as.vector(phi %*% object$d + R %*% (sqrt(object$w) *
+             object$c))
+     else as.vector(phi %*% object$d + R %*% object$c)
+ }
```

We then define a function performing one iteration of the *penalized iteratively re-weighted least squares*:

```
> gs.iter <- function(y, x, lambda, previous, transform = NULL) {
+     pred <- predict(previous, x)
+     if (!is.null(transform))
+         pred <- transform(pred)
+     w <- exp(pred)
+     u <- -y + w
+     pseudo.y <- pred - u/w
+     ss_lambdaFixed(pseudo.y, x, lambda, w)
+ }
```

We then keep our example of Sec. A.17 and extract the optimal λ_0 from the *smooth peri stimulus time histogram* fit as well. We also define some variables used in the fits at fixed λ:

```
> gfit <- evalq(gfit, env = environment(n1CitrGSSPSTH0$lambdaFct))
> Count <- gfit$mf$Count
> Time <- gfit$mf$Time
> refLambda <- 10^(gfit$nlambda)/length(Time)
> T2 <- Time/ceiling(max(Time))
> binSize <- diff(n1CitrGSSPSTH0$mids)[1]
> nbT <- n1CitrGSSPSTH0$nbTrials
> rawHist <- Count/nbT/binSize
```

We are now ready to perform fits at fixed λ. We are going to use: $\lambda = \frac{\lambda_0}{1000}, \lambda_0, 1000\,\lambda_0$. We perform 30 iterations for each. The initial guess for the first fit with $\lambda = \lambda_0$ is obtained by fitting a Gaussian regression model to the square root of the observed variable (`Count`). The two other fits use the estimate of the end of the first fit as an initial guess:

```
> f0 <- ss_lambdaFixed(sqrt(Count), T2, refLambda)
> f1 <- gs.iter(Count, T2, refLambda, f0, function(x) 0.5 *
+     log(x))
> fcurrent <- f1
> for (i in 1:29) fcurrent <- gs.iter(Count, T2,
+     refLambda, fcurrent)
> gcurrent <- fcurrent
> for (i in 1:29) gcurrent <- gs.iter(Count, T2,
+     refLambda * 1000, gcurrent)
```

```
> hcurrent <- fcurrent
> for (i in 1:29) hcurrent <- gs.iter(Count, T2,
+     refLambda/1000, hcurrent)
```

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

[6] 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.

[7] 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.

[8] John M. Chambers. *Software for Data Analysis. Programming with R.* Statistics and Computing. Springer, 2008.

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

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

[11] Peter Craven and Grace Wahba. Smoothing noisy data with spline functions. estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik*, 31:377–404, 1979. Available at: http://resolver.sub.uni-goettingen.de/purl?GDZPPN00117486X.

[12] David J. Cummins, Tom G. Filloon, and Douglas Nychka. Confidence intervals for nonparametric curve estimates: Toward more uniform pointwise coverage. *Journal of the American Statistical Association*, 96(453):233–246, mar 2001.

[13] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986. Available at: http://cg.scs.carleton.ca/~luc/rnbookindex.html.

[14] Ricardo Escola, Christophe Pouzat, Antoine Chaffiol, Blaise Yvert, Isabelle E. Magnin, and Regis Guillemaud. Simone : A neural simulator to test mea-embedded algorithms. *IEEE Transactions on Neural and Rehabilitation Engineering*, 16(2):149–160, apr 2008.

[15] Robert Gentleman and Duncan Temple Lang. Statistical Analyses and Reproducible Research. Working Paper 2, Bioconductor Project Working Papers, 29 May 2004. Available at: http://www.bepress.com/bioconductor/paper2/.

[16] 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.

[17] Chong Gu. Cross-validating non-gaussian data. *Journal of Computational and Graphical Statistics*, 1(2):169–179, jun 1992.

[18] Chong Gu. *Smoothing Spline Anova Models*. Springer, 2002.

[19] Chong Gu. Smoothing noisy data via regularization: statistical perspectives. *Inverse Problems*, 24(3):034002–, 2008.

[20] Chong Gu and Dong Xiang. Cross-validating non-gaussian data: Generalized approximate cross-validation revisited. *Journal of Computational and Graphical Statistics*, 10(3):581–591, sep 2001.

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

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

[23] Karl-Ernst Kaissling. *R H Wright Lectures on Insect Olfaction*. Munich: Typographischer Betrieb, W Biering, H Numberger, 1987.

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

[25] 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.

[26] George Kimeldorf and Grace Wahba. Some results on tchebycheffian spline functions. *J. Mathematical Analysis and Applications*, 33(1):82–95, 1971. Available at: http://www.stat.wisc.edu/~wahba/ftp1/oldie/kw71.pdf.

[27] 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.

[28] Pierre L'Ecuyer and Josef Leydold. rstream: Streams of random numbers for stochastic simulation. *R News*, 5(2):16–20, November 2005. Available from: http://CRAN.R-project.org/doc/Rnews/.

[29] Pierre L'Ecuyer, Richard Simard, E. Jack Chen, and W. David Kelton. An objected-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075, 2002. Available at: http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf.

[30] Joseph Leydold. *rstream: Streams of random numbers*, 2007. R package version 1.2.2.

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

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

[33] Paul Murrell. Introduction to data technologies. Available at: http://www.stat.auckland.ac.nz/~paul/ItDT/, aug 2008.

[34] Douglas Nychka. Bayesian confidence intervals for smoothing splines. *Journal of the American Statistical Association*, 83(404):1134–1143, dec 1988.

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

[36] Roger D. Peng. *cacheSweave: Tools for caching Sweave computations*, 2007. R package version 0.4-3.

[37] 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.

[38] 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.

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

[40] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[41] Anthony Rossini, Luke Tierney, and Na Li. Simple Parallel Statistical Computing in R. UW Biostatistics Working Paper Series 193, University of Washington, 2003. Available from: http://www.bepress.com/uwbiostat/paper193/.

[42] Luke Tierney, A. J. Rossini, Na Li, and H. Sevcikova. *snow: Simple Network of Workstations*. R package version 0.3-3.

[43] 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.

[44] 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.

[45] Grace Wahba. Bayesian "confidence intervals" for the cross-validated smoothing spline. *Journal of the Royal Statistical Society. Series B (Methodological)*, 45(1):133–150, 1983.

[46] Grace Wahba. *Spline Models for Observational Data*. SIAM, 1990.

[47] Garrick Wallstrom, Jeffrey Liebner, and Robert E. Kass. An Implementation of Bayesian Adaptive Regression Splines (BARS) in C with S and R Wrappers. *Journal of Statistical Software*, 26(1):1–21, 2 2007.

[48] Dong Xiang and Grace Wahba. A generalized approximate cross validation for smoothing splines with non-gaussian data. *Statistica Sinica*, 6:675–692, 1996. Available at: http://www3.stat.sinica.edu.tw/statistica/j6n3/j6n312/j6n312.htm.