FRETBursts: Open Source Burst Analysis Toolkit for Confocal Single-Molecule FRET

Antonino Ingargiola*[1], Eitan Lerner[1], SangYoon Chung[1], Shimon Weiss[1], and Xavier Michalet[1]

[1]Dept. Chem. & Biochem, Univ. California Los Angeles, Los Angeles, CA, USA.

February 8, 2016

Abstract

Single-molecule Förster Resonance Energy Transfer (smFRET) allows probing intermolecular interactions and conformational changes in biomacromolecules, and represents an invaluable tool in studying cellular processes at the molecular scale [21]. smFRET experiments can detect the distance between two fluorescent labels (*donor* and *acceptor*) in the 3-10 nm range. In the commonly employed confocal geometry, molecules are free to diffuse in solution. When a molecule traverses the excitation volume it emits a burst of photons that can be detected by single-photon avalanche detectors (SPADs). The intensities of donor and acceptor fluorescence can then be related to the distance between the two dyes.

The analysis of smFRET experiments involves identifying photon bursts from single-molecules in a continuous stream of photon, estimating the background and other correction factors, filtering and finally extracting the corrected FRET efficiencies for each sub-population in the sample. In this paper we introduce FRETBursts, a software for confocal smFRET data analysis which allows executing a complete burst analysis pipeline by using state-of-the-art algorithms. FRETBursts is an open source python package that we envision both as toolkit for research and new developments in burst analysis and as reference implementation of commonly employed algorithms. We follow the highest standard in software development to ensure that the source is easy to read, well documented and thoroughly tested. Moreover, in an effort to lower the barriers to computational reproducibility, we embrace a modern workflow based on Jupyter notebooks that allows to capture of the whole process from raw data to figures within a single document.

Contents

1 Introduction

1.1 Open Science and Reproducibility

In the last 20 years, single molecule FRET (smFRET), has emerged as one of the most useful techniques in single-molecule spectroscopy [16,43]. Except a few specific ensem-

*ingargiola.antonino@gmail.com

ble time-resolved measurements [24, 33], smFRET unique feature is the ability to resolve conformational changes of biomolecules or measure binding-unbinding kinetics on heterogeneous samples. smFRET measurements on freely diffusing molecules (the focus of this paper) have the advantage of probing molecules and processes without possible perturbation from surface immobilization [5, 8].

The field of freely-diffusing smFRET data analysis, has seen a number of significant contributions over the years [1, 4, 7, 10–13, 23, 29, 35, 38, 39, 45]. However, so far, except for some fundamental steps of burst analysis, there is no single approach that is universally accepted and broadly applied. On one hand this situation stems from the fact that different approaches tend to answer slightly different questions. On the other hand, this is the result of the trade off in each research lab between accuracy and software complexity, in particular on the amount of effort each group wants to invest in implementing new non-trivial methods reported in literature.

In fact, except for a recent laudable example [26], each research group have reimplemented its own unpublished or closed-source versions of the analysis software, with very little collaboration or code sharing. Even in our group, past smFRET papers merely mention the usage of custom-made software without additional details [23, 29]. The fact that setups for freely-diffusing smFRET can significantly vary (in number of polarization or spectral channels for example), makes the problem only worst. This situation, represents a real impediment to the scientific progress because:

(a) As new methods are proposed in literature, understandably, only few groups are willing (or have the resources) to invest the time necessary to re-implement them in their own software, especially when contributions are incremental. In the rare occasions of new methods being groundbreaking, the reimplementation requires a systematic duplication of efforts.

(b) Both the non-availability of the software and the differences in implementation details render a direct comparison of different software implementations difficult, if not impossible. This limits the ability to cross-validate the correctness of different implementations of the same method or to compare accuracy and effectiveness of different methods.

(c) The variety of optical setups resulted in an accumulation of different incompatible file binary formats, some vendor-specific and some custom-made. This clearly represent an additional barrier in data exchange between groups using different hardware and aggravate the previous issues.

Effectively, these issues have created islands of non-communicating groups (regarding software and data sharing), in stark contrast with the consolidated tradition, in the biological sciences, of openly sharing reagents and samples.

From a more general stance, since the pioneering work of Donoho group in the 90's [3], became clear that developing and maintaining open source scientific software for reproducible research is a critical requirement for progress in modern science [17, 41]. Peer-reviewed publications describing such software are also necessary [31], although the debate is still open on the most effective model for peer-reviewing this class of publications [14, 15] (Willson 2015) (Mills 2015) (Brown 2015 and 2013).

Facing the previous issues, we decided to develop FRETBursts, an open source Python software for burst analysis of freely-diffusing single-molecule FRET experiments. With FRETBursts we provide a tool that is available to any scientist to use, study and modify. Furthermore, FRETBursts execution model based on Juyter Notebook [36] is designed to facilitate computational reproducibility. FRETBursts is hosted and openly developed on GitHub [2, 32], where users can send comments, report issues or contribute code. In a parallel effort, some of the authors recently introduced a common file format for timestamp-based single-molecule fluorescence experiments called Photon-HDF5 [18]. The work on Photon-HDF5, which is fully supported by FRETBursts, complements the ecosystem of open tools for reproducible science in the single-molecule field.

Understanding smFRET burst analysis requires several concepts and definitions. In this paper we aim to provide a brief introduction to smFRET analysis concepts and terminology used by FRETBursts. We illustrate how to perform the fundamental steps of burst analysis, highlighting key parameters and algorithms available. The aim is not covering all FRETBursts features and options but providing an overview detailed enough for starting using FRETBursts and customizing the analysis. For additional information, we refer the reader to the FRETBursts Reference Documentation (link) and to the FRETBursts μs-ALEX notebook (link) Finally, usage questions can be posted by opening an Issue on GitHub (link).

1.2 Paper Overview

The paper is structured as follows. In the next section 2 we give an overview of the software features, the modality of execution and the development style. In section 3, we review a few preliminary concepts and terminology needed to understand the smFRET burst analysis. In section 4, we detail the execution the main steps involved in smFRET burst analysis: data loading (section 4.1), defining excitation alternation periods (section 4.2), background correction (section 4.3), burst search (section 4.4), burst selection (section 4.5) and FRET fitting (section 4.6). The aim is to provide the reader with enough information to understand the specificities of the different algorithms and to be able to adapt the analysis to new situations. In section 5, we show how to implement a new burst analysis in FRETBursts, taking as an example the Burst Variance Analysis (BVA) [39]. In section 6 we describe the development process, the testing infrastructure and contributors guidelines. Finally, in section 7, we summarize what we believe to be the strengths of FRETBursts software.

Throughout this paper, links to relevant sections of documentation and other web resources are displayed as "(link)".

In order to make the text accessible to the widest number of readers, we concentrated python-specific details in special subsections titled *Python details*. These subsections provide deeper insights for readers already familiar with python and can be safely skipped otherwise. Finally, note that all commands here reported can be found in the accompanying notebooks (link).

2 FRETBursts Overview

2.1 Features

FRETBursts can analyze smFRET measurements from one or multiple excitation spots [19]. The supported excitation schemes include single laser, alternating laser excitation (ALEX) with either CW lasers (µs-ALEX [20]) or pulsed lasers (ns-ALEX [22] or pulsed-interleaved excitation (PIE) [27]).

The software implements both standard and novel algorithms for smFRET data analysis including background estimation as a function of time (including background accuracy metrics), sliding-window burst search [8], dual-channel burst search (DCBS) [29] and modular burst selection methods based on user-defined criteria (including a large set of pre-defined selection rules). Novel features include burst size selection with γ-corrected burst sizes, burst weighting, burst search with background- dependent threshold (in order to guarantee a minimal single-to background ratio [25]). Moreover, FRETBursts provides a large set of fitting options to characterize FRET populations. In particular, distributions of burst quantities (such as E or S) can be assessed through (1) histogram fitting (with arbitrary model functions), (2) non-parametric weighted kernel density estimation (KDE), (3) weighted expectation-maximization (EM), (4) maximum likelihood fitting using Gaussian models or Poisson statistic. Finally FRETBursts includes a large number of predefined and customizable plot functions which (thanks to the *matplotlib* graphic library) produce publication quality plots in a wide range of formats.

2.2 Notebook Workflow

FRETBursts has been developed with the goal of facilitating computational reproducibility of the performed data analysis [3]. For this reason, the preferential way of using FRETBursts is by executing one of the tutorials which are in the form of Jupyter notebooks [36]. Jupyter (formerly IPython) notebooks are web-based documents which contain both code and rich text (including equations, hyperlinks, figures, etc...). FRETBursts tutorials are notebooks which can be re-executed, modified or used to process new data files with minimal modifications. The "notebook workflow" [36] not only facilitates the description of the analysis (by integrating the code in a rich document) but also greatly enhance its reproducibility by storing an execution trail that includes software versions, input files, parameters, commands and all the analysis results (text, figures, tables, etc...).

The Jupyter Notebook environment streamlines FRETBursts execution (compared to a traditional script and terminal based approach) and allows FRETBursts to be used even without prior python knowledge. The user only needs to get familiar with the notebook graphical environment, in order to be able to navigate and run the notebooks. The list of FRETBursts notebooks can be found in the `FRETBursts_notebooks` repository on GitHub (link).

2.3 Development Model

FRETBursts (and the entire python ecosystem it depends on) is open source and the source code is fully available for any scientist to study, review and modify. The authors encourage users to use GitHub issues for questions, discussions and bug reports, and to submit patches through GitHub pull requests.

In order to minimize the likelihood of bugs and erroneous results, FRETBursts is developed following modern software engineering techniques such as defensive programming, unit testing, regression testing and continuous integration [44].

The open source nature of FRETBursts and of the python ecosystem, not only makes it a more transparent, reviewable platform for scientific data analysis, but also allows to leverage state-of-the-art online services as GitHub (link) for hosting, issues tracking and code reviews, TravisCI (link) for continuous integration (i.e. automated test suite execution on multiple platforms after each commit) and ReadTheDocs.org for automatic documentation building and hosting. All these services would be extremely costly, if available *tout court*, for a proprietary software or platform [9].

3 Architecture and Concepts

In this section we introduce some general concepts and naming conventions related to the smFRET burst analysis in FRETBursts.

3.1 Photon Streams

The fundamental data at the core of smFRET experiments is the array of photon arrival timestamps, with a temporal resolution set by the acquisition hardware, ranging from below nanoseconds to a few tens of nanoseconds. In single-spot measurements, all timestamps are stored in a single array. In multi-spot measurements [19], there are as many timestamps arrays as excitation spots.

Each array contains timestamps from both donor (D) and acceptor (A) channels. In ALEX measurements [23], we can further differentiate between photons emitted during D and A excitation periods. In FRETBursts the different selections of photons/timestamps are called "photon streams" and they are specified with a `Ph_sel` object (link). In non-ALEX smFRET data there are 3 photon streams (table 1), while in ALEX data we have 5 base photon streams (table 2).

Photon selection	code
All-photons	`Ph_sel('all')`
D-emission	`Ph_sel(Dex='Dem')`
A-emission	`Ph_sel(Dex='Aem')`

Table 1: Photon selection syntax (non-ALEX)

Photon selection	code
All-photons	`Ph_sel('all')`
D-emission during D-excitation	`Ph_sel(Dex='Dem')`
A-emission during D-excitation	`Ph_sel(Dex='Aem')`
D-emission during A-excitation	`Ph_sel(Aex='Dem')`
A-emission during A-excitation	`Ph_sel(Aex='Aem')`

Table 2: Photon selection syntax (ALEX)

The `Ph_sel` class (link) allows the expression of any combination of photon streams. For example, in ALEX measurements, the D-emission during A-excitation stream is usually excluded because it does not contain any useful signal [23]. To indicate all but the photons in this photon stream we write `Ph_sel(Dex='DAem', Aex='Aem')`, which indicates selection of donor and acceptor photons (`DAem`) during donor excitation (`Dex`) and only acceptor photons (`Aem`) during acceptor excitation (`Aex`).

3.2 Background Definitions

An estimation of the background rates is needed both to select a proper threshold for burst search and to correct the raw burst counts by subtracting the background counts.

The recorded stream of timestamps is the result of two processes: one characterized by a high count rate, due to fluorescence photons of single molecules crossing the excitation volume, and another one characterized by a lower count rate due to "background counts" originating from the detectors dark counts, out of focus molecules and sample scattering and/or auto-fluorescence [12]. The signature of those two processes can be observed in the distribution of timestamp delays (i.e. the waiting times between two subsequent timestamps) as illustrated in figure 2(a). The "tail" of the distribution (a straight line in semi-log scale) corresponds to exponentially-distributed delays, indicating that those counts are generated by a Poisson process (link). At short timescales, the distribution departs from exponential behavior due to the contribution of the higher rate process of single molecules traversing the excitation volume. To estimate the background rate, (i.e. the exponential time constant) it is necessary to define a delay threshold, above which the distribution can be considered exponential. Next a fitting method, for example the Maximum Likelihood Estimation (MLE) or a curve fit of the histogram via non-linear least squares (NLSQ) must be selected.

It is advisable to check the background at different time points throughout the measurements in order to track possible variations. Experimentally, we found that when the background is not constant, it usually varies on time scales of tens of seconds (see figure 3). FRETBursts splits the data in uniform time slices called *background periods* and computes the background rates for each of these slices (see section 4.3). Note that the the same splitting in background periods is used during burst search to compute a background-dependent threshold and to apply the burst correction (section 4.4).

Name	Description
`nd`	number of photons detected by the donor channel (during donor excitation period in ALEX case)
`na`	number of photons detected by the acceptor channel (during donor excitation period in ALEX case)
`naa`	number of photons detected by the acceptor channel during acceptor excitation period (present only in ALEX measurements)

Table 3: `Data` attributes names and descriptions for burst photon counts in different photon streams.

3.3 The `Data` Class

The `Data` class (link) is the fundamental data container in FRETBursts. It contains the measurement data and parameters (attributes) as well as several methods for data analysis (background estimation, burst search, etc...). All analysis results (bursts data, estimated parameters) are also stored as `Data` attributes.

There are 3 important "burst counts" attributes which contains the number of photon detected in donor or acceptor channel during donor or acceptor excitation (table 3). The attributes in table 3 are background-corrected by default. Furthermore, `na` is corrected for leakage and direct excitation (section 4.4.2) if the relative coefficients are specified (by default they are 0). There is also a closely related attribute named `nda` for donor photons during acceptor excitation. `nda` is normally neglected as it only contains background.

Python details Many `Data` attributes are list of arrays (or scalars) with list-length equal to the number of excitation spots. This means that, in single-spot measurements, to access an array of burst-data we always have to specify the index 0, for example `Data.nd[0]`. `Data` implements a shortcut syntax to access the first element of a list with an underscore, so we can type equivalently use `Data.nd_` instead of `Data.nd[0]`.

3.4 Introduction to Burst Search

Identifying single-molecule bursts in the stream of photons is one of the most crucial steps in the analysis of freely-diffusing single-molecule FRET data. The widely used "sliding window" algorithm, introduced by the Seidel group in

1998 ([8], [10]), involves searching for m consecutive photons detected during a period shorter than Δt. In other words, bursts are regions of the photon stream where the local rate (computed using m photons) is above a minimum threshold rate. Since a universal criterion to choose the rate threshold and the number of photons m is, as of today, lacking, it has become a common practice to manually adjust those parameters for each specific measurement.

A more general approach consists in taking into account the background rate of the specific measurements and in choosing a rate threshold that is F times larger than the background rate. This approach assures that all the resulting bursts have a signal-to-background ratio (SBR) larger than $(F - 1)$ [25]. A consistent criterion for choosing the threshold is particularly important when comparing different measurements with different background rates, when the background significantly varies during measurements or in multi-spot measurements where each spot has a different background rate.

A second important aspect of burst search is the choice of photon stream used to perform the search. In most cases, for instance when identifying FRET populations, the burst search should use all photons (i.e. ACBS). In some other cases, when focusing on donor-only or acceptor only populations, it is better to perform the search using only donor or acceptor signal. In order to handle the general case and to provide flexibility, FRETBursts allows to perform the burst search on arbitrary selections of photons. (see section 3.1 for more info on photon stream definitions).

Additionally, Nir *et al.* [29] proposed a DCBS, which can help to mitigate artifacts due to photo-physical effects such as blinking. In DCBS a search is performed independently on two photon streams and bursts are marked only when both photon streams exhibit a rate higher than the threshold, implementing an AND-gate logic. Usually, the term DCBS refers to a burst search where the two photon streams are (1) all photons during donor excitation (`Ph_sel(Dex='DAem')`) and (2) acceptor channel photons during acceptor excitation (`Ph_sel(Aex='Aem')`).

After each burst search, it is necessary to select bursts having a minimum number of photons (burst size). In the most basic form, this selection can be performed during burst search by discarding bursts with size smaller than a threshold L, as originally proposed by Eggeling *et al.* [8]. This method, however, neglects the effect of background and γ factor on the burst size and can lead to a selection bias of certain channels and/or sub-populations. For this reason we encourage performing a burst size selection after background correction, possibly taking into account the γ factor, as discussed in sections 3.5 and 4.5. In special situations, users can also choose to replace this size selection with another criterion such as burst duration or brightness.

3.5 γ-corrected Burst Sizes and Weights

The number of photons detected during a burst, commonly called "burst size", is usually computed using either all photons, or photons detected during donor excitation period. To compute the burst size, FRETBursts uses one of the following formulas:

$$n_{dex} = n_a + \gamma\, n_d \tag{1}$$

$$n_t = n_a + \gamma\, n_d + n_{aa} \tag{2}$$

where n_d, n_a and n_{aa} are, similarly to the attributes in table 3, the background-corrected burst counts in different channels and excitation periods. γ, called the "γ factor", takes into account different quantum yields of donor and acceptor dyes and different photon detection efficiencies between donor and acceptor detection channels [23]. Eq. 1 includes only photons during donor excitation periods, while eq. 2 includes all photons. Burst sizes computed according to eq. 1 or 2 are called γ-corrected burst sizes.

The burst search algorithm yields a set of bursts whose sizes approximately follows an exponential distribution. Bursts with large sizes (which contain most of the information) are much less frequent than bursts with smaller sizes. For this reason, it is important to select burst sizes larger than a threshold in order to properly characterize FRET populations (see section 4.5).

Selecting bursts by size is a critically important step. A too low threshold will broaden the FRET populations and introduce artifacts (spurious peaks and patterns) due to the majority of bursts having E and S computed from ratios of small integers. Conversely, a too high threshold will result in a lower number of bursts and possibly poor statistics in representing FRET populations. Additionally, when selecting bursts (see section 4.5), it is important to use γ-corrected burst sizes, in order to avoid under-representing some FRET sub-populations due to different quantum yields between donor and acceptor dyes and/or different photon detection efficiencies of donor and acceptor emission.

A simple way to mitigate the dependence on the burst size threshold is weighting bursts according to their size (i.e. their information content) so that the bursts with largest sizes will have the largest weights. The weighting can be used to build weighted histograms or Kernel Density Estimation (KDE) plots. When using weights, the choice of a particular burst size threshold affects the shape of the burst distribution to a lesser extent, therefore lower thresholds can be used (yielding to better statistics) without broadening the peaks of sub-populations (yielding to better population identification).

Python details FRETBursts has the option to weight bursts using γ-corrected burst sizes which optionally include acceptor excitation photons `naa`. A weight proportional to the burst size is applied by passing the argument `weights='size'` to histogram or KDE plot functions. The weights keyword can be also passed to fitting functions in order to fit the weighted E or S distributions (see section 4.6). Several other weighting functions (for example quadratical) are listed in the fret_fit.get_weights documentation (link).

3.6 Plotting Data

FRETBursts uses matplotlib [6] and seaborn [42] to provide a wide range of built-in plot functions (link) for Data objects. The plot syntax is the same for both single and multi-spot measurements. The majority of plot commands are called through the wrapper function dplot, for example to plot a timetrace of the photon data, type:

```
dplot(d, timetrace)
```

The function dplot is the generic plot function which creates figure and handles details common to all the plotting functions (for instance the title). d is the Data variable and timetrace is the actual plot function which operates on a single channel. In multi-spot measurements dplot creates one subplot for each spot and calls timetrace for each channel.

All built-in plot functions which can be passed to dplot are defined in the burst_plot module (link).

Python details When FRETbursts is imported, all plot functions are also imported. To facilitate finding the plot functions through auto-completion, their names start with a standard prefix indicating the plot type. The prefixes are: timetrace for binned timetraces of photon data, ratetrace for rates of photons as a function of time (non binnned), hist for functions plotting histograms and scatter for scatter plots. Additional plots can be easily created directly with matplotlib.

By default, in order to speed-up batch processing, FRETBursts notebooks display plots as static images using the *inline* matplotlib backend. User can switch to interactive figures inside the browser by activating the interactive backend with the command %matplotlib notebook. Another option is displaying figures in a new standalone window using a desktop graphical library such as QT4. In this case the command to be used is %matplotlib qt.

A few plot functions such as timetrace and hist2d_alex have interactive features which require the QT4 backend. As an example, after switching to the QT4 backend the following command:

```
dplot(d, timetrace, scroll=True, bursts=True)
```

will open a new window with a timetrace plot with overlay of bursts, and an horizontal scroll-bar for quick "scrolling" throughout time. The user can click on a burst to have the corresponding burst info be printed in the notebook. Similarly, calling the hist2d_alex function with the QT4 backend allows selecting an area on the E-S histogram using the mouse.

```
dplot(ds, hist2d_alex, gui_sel=True)
```

The values that identify the region are printed in the notebook and can be passed to the function select_bursts.ES to select bursts inside that region (see section 4.5).

4 smFRET Burst Analysis

4.1 Loading the Data

While FRETBursts can load data files from a few file formats, the authors promote Photon-HDF5 [18], an HDF5-based open format specifically designed for freely-diffusing smFRET and other timestamp-based experiments. Photon-HDF5 is a self-documented platform and language independent binary format which support compression and allows saving photon-data (e.g. timestamps) and measurement-specific meta-data (setup and sample information, authors, provenance etc...). Moreover, Photon-HDF5 is designed for long-term data preservation and aims to facilitate data sharing among different software and research groups. FRETBursts example data files are in Photon-HDF5 format and can be opened with stand-alone viewers (such as HDFView, link) or programming language.

To load data from a Photon-HDF5 file we use the function loader.photon_hdf5 (link) as follows:

```
d = loader.photon_hdf5(file_name)
```

where file_name is a string containing the file path. This command loads all the measurement data into the variable d, a Data object (see section 3.3).

The same command can load data from a variety of measurement types stored in a Photon-HDF5 file. For instance, data generated using different excitation schemes (CW vs pulsed, single-laser vs 2 alternating lasers) or with any number of excitation spots is automatically recognized.

Other file formats which FRETBursts can load include µs-ALEX data stored in SM format (a custom binary format used in S.W. lab), ns-ALEX data stored in SPC format (a binary format used by TCSPC Becker & Hickl cards). ns-ALEX data in HT3 format (a binary format used by PicoQuant hardware) can be easily converted to Photon-HDF5 using the phconvert converter (link) and then loaded in FRETBursts. More information on loading these file formats and on manually loading other arbitrary formats can be found in the loader module's documentation (link).

4.2 Alternation Parameters

In case of µs-ALEX and ns-ALEX data, it is necessary to define the alternation periods for donor and acceptor excitation. In µs-ALEX measurements, CW lasers are alternated on timescales of 10-100 µs. By plotting the histogram of the timestamps modulo the alternation period is possible to identify the donor and acceptor periods (see figure 1a). In ns-ALEX measurements, pulsed lasers are interleaved with typical separation of 10-100 ns. In this case the histogram

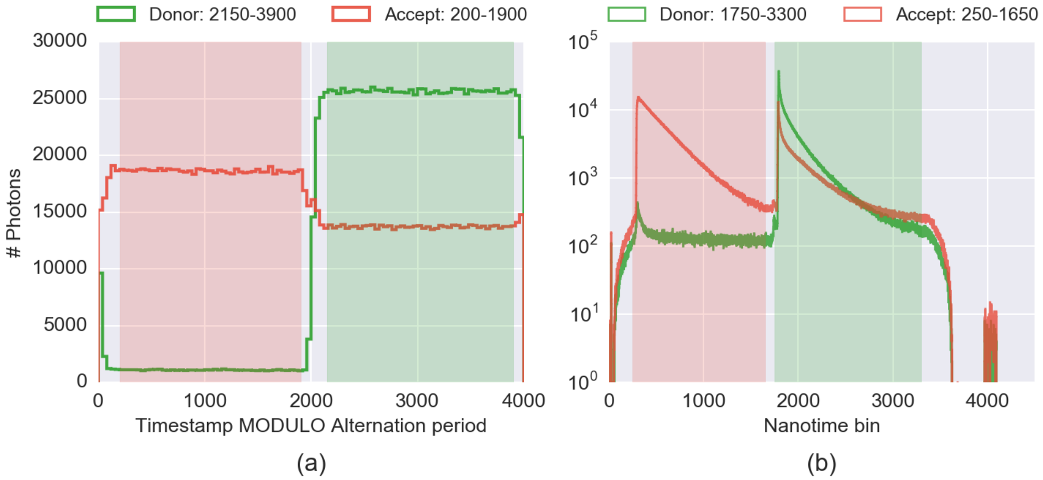

Figure 1: Histograms used for the selection/determination of the alternation periods for two typical smFRET-ALEX experiments. Distributions of photons detected by the donor channels are in *green*, and by the acceptor channel in *red*. The light *green* and *red* shaded areas mark the donor and acceptor period definitions. (a) μs-ALEX alternation histogram, i.e. histogram of timestamps *modulo* the alternation period for a smFRET sample. (b) ns-ALEX nanotime histogram for a smFRET sample. Both plots have been generated by the same plot function (`plot_alternation_hist()`). Additional information on these specific measurements can be found in the attached notebook.

of the TCSPC nanotimes will allow the definition of the period of fluorescence after excitation of either the donor or the acceptor (see figure 1b).

In both cases, the functions `plot_alternation_hist` (link) will plots the relevant alternation histogram (figure 1) using currently selected (or default) values for donor and acceptor excitation periods.

To change the period definitions, the user can type:

```
d.add(D_ON=(2850, 580), A_ON=(900, 2580))
```

where `D_ON` and `A_ON` are tuples (pairs of numbers) representing the *start* and *stop* values for D or A excitation periods. The previous command works both for μs-ALEX and ns-ALEX measurements.

After changing the parameters, a new alternation plot will show the updated period selections.

When the alternation period definition is correctly defined, it can be applied using the function `loader.alex_apply_period` (link):

```
loader.alex_apply_period(d)
```

After this command, `d` will contain only photons inside the defined excitation periods. At this point, in order to further change the period definitions, it is necessary to reload the data file.

4.3 Background Estimation

The first step of smFRET analysis involves estimating background rates. For example, to compute the background every 30 s, using a minimal inter-photon delay threshold of 2 ms for all the photon, we use:

```
d.calc_bg(bg.exp_fit, time_s=30, tail_min_us=2000)
```

The first argument (`bg.exp_fit`) is the underlying function used to fit the background in each period and for each photon stream (see section 3.2). The function `bg.exp_fit` estimates the background using a maximum likelihood estimation (MLE) of the delays distribution. Additional fitting functions are available in `bg` namespace (i.e. the `background` module, link). The second argument, `time_s`, is the *background period* (section 3.2) and the third, `tail_min_us`, is the inter-photon delay threshold above which the distribution is assumed exponential. It is possible to use different thresholds for each photon stream, passing a tuple (i.e. a comma-separated list of values, link) instead of a scalar. Finally, it is possible to use a heuristic estimation of the threshold using `tail_min_us='auto'`. For more details refer to the `Data.calc_bg` documentation (link).

FRETBursts provides are two kind of plots to represent the background. One is the histograms of inter-photon delays compared to the fitted exponential distribution reported in figure 2) (see section 3.2 for details on the inter-photon distribution). This plot is performed with the command:

```
dplot(d, hist_bg, bp=0)
```

The argument `bp` is an integer specifying the background period to be plotted. When not specified the default is 0, i.e. the first period. Figure 2 allows to quickly identify pathological cases when the background fitting procedure returns unreasonable values.

The second background-related plot is a timetrace of background rates, as shown in figure 3. This plot allows to monitor background changes taking place during the measurement and is obtained with the command:

```
dplot(d, timetrace_bg)
```

Normally, samples should have a constant background as a function of time like in figure 3(a). However, oftentimes, non-ideal experimental conditions can yield a time-varying background, as shown in figure 3(b). For example, when the sample is not sealed in an observation chamber, evaporation can induce background variations (typically increasing) as a function of time. Additionally, cover-glass impurities can contribute to the background even when focusing deep into the sample (10µm or more). These impurities tend to bleach on timescales of minutes resulting in background variations during the course of the measurement.

Python details For an ALEX measurement, the tuple passed to `tail_min_us` to define the thresholds, is required to have have 5 values corresponding the 5 photon streams. The order of the photon streams can be obtained from the `Data.ph_streams` attribute (i.e. `d.ph_streams` in our example). The estimated background rates are stored in the `Data` attributes `bg_dd`, `bg_ad` and `bg_aa`, corresponding to the photon streams `Ph_sel(Dex='Dem')`, `Ph_sel(Dex='Aem')` and `Ph_sel(Aex='Aem')` respectively. These attributes are lists of arrays (one array per excitation spot). The arrays contain the estimated background rates in the different background periods.

4.3.1 Error Metrics and Optimal Threshold

The functions used to fit the background provide also a goodness-of-fit estimator computed on the basis of the empirical distribution function (EDF) [30, 37]. The "distance" between the EDF and the theoretical (i.e. exponential) cumulative distribution represents and indicator of the quality of fit. Two different distance metrics can be returned by the background fitting functions. The first is the Kolgomorov-Smirnov statistics, which uses the maximum of the difference between the EDF and the theoretical distribution. The second is the Cramér von Mises statistics corresponding to the integral of the squared residuals (see the code for more details, link).

In principle, the optimal inter-photon delay threshold will minimize the error metric. This approach is implemented by the function `calc_bg_brute` (link) which performs a brute-force search in order to find the optimal threshold. This level of sophistication in estimating the background rates is not necessary under typical experimental conditions, as the difference between an optimal threshold and a manually (or heuristically) chosen one will be small if not negligible in most practical cases.

4.4 Burst Search

4.4.1 Burst Search in FRETBursts

Following background estimation, burst search is the next step of the analysis. In FRETBursts, a standard burst search on a single photon stream (see section 3.4) is performed by calling the `Data.burst_search` method (link). For example, the following command:

```
d.burst_search(F=6, m=10, ph_sel=Ph_sel('all'))
```

performs a burst search on all photons (`ph_sel=Ph_sel('all')`), with a minimum rate 6 times larger than the background rate (`F=6`) and using 10 consecutive photons to compute the local rate (`m=10`). A different photon selection, threshold (F) or number of photons for rate computation m can be selected by passing a different value. These parameters are generally a good starting point for smFRET analysis but can be adjusted in specific cases.

Note that, in the previous burst search, no burst size selection was performed (i.e. the minimum bursts size is effectively m). An additional parameter L can be passed to apply a threshold on the raw burst size (before any correction). It is recommended, however, to select bursts only after the background correction is applied as shown in the next section 4.5.

It might sometimes be useful to specify a fixed photon-rate threshold, instead of a threshold depending on the background rate, as in the previous example. In this case, instead of F, the argument `min_rate_cps` can be used to specify the threshold (in Hz). For example, a burst search with a 50 kHz threshold can be performed as follows:

```
d.burst_search(min_rate_cps=50e3, m=10,
               ph_sel=Ph_sel('all'))
```

Finally, to perform a DCBS burst search (or in general an AND gate burst search, see section 3.4) we use the function `burst_search_and_gate` (link) as in the following example:

```
d_dcbs = bext.burst_search_and_gate(d, F=6, m=10)
```

The last command puts the burst search results in a new copy the `Data` variable *d* (the copy is here called `d_dcbs`). Since FRETBursts shares the arrays timestamps and detectors between different copies of `Data` objects, the memory usage is contained even when using several copies.

Python details Note that, while `.burst_search()` is a method of `Data`, `burst_search_and_gate` is a function in the `bext` module taking a `Data` object as a first argument and returning a new `Data` object.

The function `burst_search_and_gate` accepts optional arguments, `ph_sel1` and `ph_sel2`, whose default values correspond to the classical DCBS photon stream selection (see section 3.4). These arguments can be specified to select different photon streams than in a classical DCBS.

The `bext` module (link) collects "plugin" functions that provides additional algorithms for processing `Data` objects.

4.4.2 Correction Coefficients

In µs-ALEX there are 3 important correction parameters: γ-factor, donor spectral leakage into the acceptor channel and acceptor direct excitation by the donor excitation laser [23]. These corrections can be applied by simply assigning to the respective `Data` attributes:

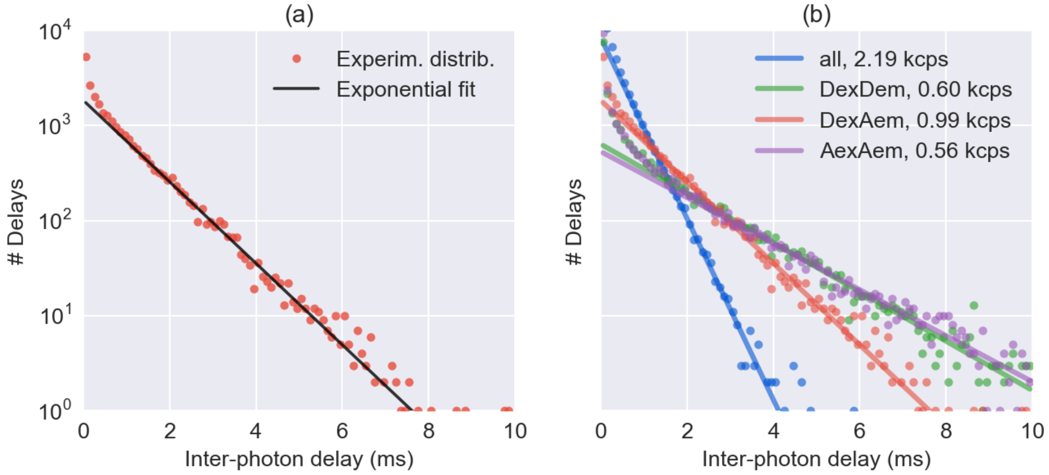

Figure 2: Experimental distributions of inter-photon delays (*dots*) and corresponding fits of the exponential tail (*solid lines*). (*Panel a*) An example of inter-photon delays distribution (*red dots*) and an exponential fit of the tail of the distribution (*black line*). (*Panel b*) Inter-photon delays distribution and exponential fit for different photon streams as obtained with `dplot(d, hist_bg)`. The *dots* represent the experimental histogram for the different photon streams. The *solid lines* represent the corresponding exponential fit of the tail of the distributions. The legend shows abbreviations of the photon streams and the fitted rate backround rate.

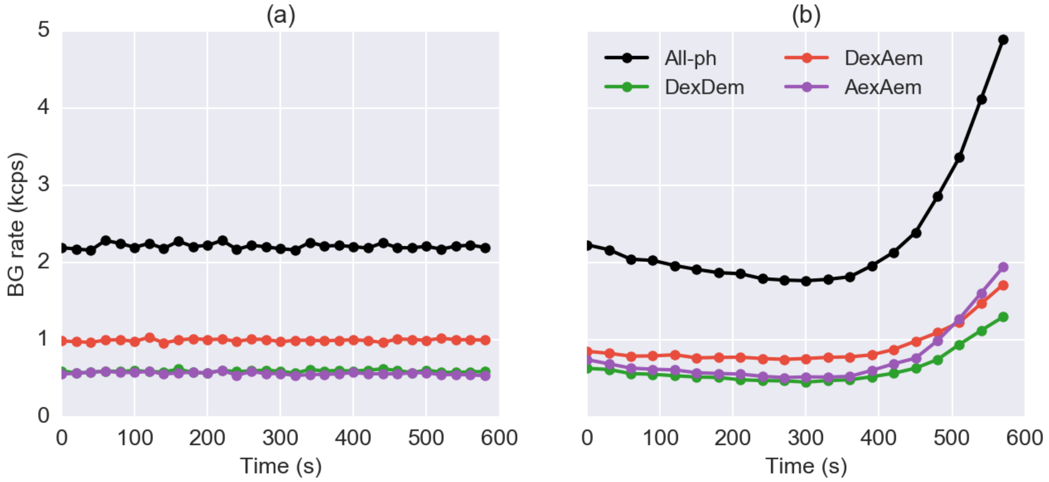

Figure 3: Estimated background rate as a function of time for two μs-ALEX measurements. Different colors represent different photon streams. (*Panel a*) A measurement performed with a sealed sample chamber exhibiting constant a background as a function of time. (*Panel a*) A measurement performed on an unsealed sample exhibiting significant background variations due to sample evaporation an to photo-bleaching of impurities during the measurements (likely on the coverglass). These plots are produced by the command `dplot(d, timetrace_bg)` after estimation of background. Each data point in these figures is computed for a 30 s time window.

```
d.gamma = 0.85
d.leakage = 0.04
d.dir_ex = 0.08
```

These attributes can be assigned either before or after the burst search. In the latter case, existing burst data is automatically updated using the new correction parameters.

These correction factors can be used to display corrected FRET distributions. However, in order to resolve the peak FRET efficiency value, we found that is more accurate to fit the FRET histogram without those corrections (i.e. background-corrected proximity ratio). Next, we can use an algebraic formula to correct fitted peak positions (see SI of [23]) and obtain the corrected FRET efficiency. FRETBursts implements the correction formulas for E and S in the functions `fretmath.correct_E_gamma_leak_dir` and `fretmath.correct_S` (link). A complete derivation of all these correction formulas (and their inverse) has been posted as a notebook (link).

4.5 Burst Selection

After burst search, it is common to select bursts according to different criteria. One of the most common is the burst size.

For instance, to select bursts with more than 30 photons (computed after background correction) detected during the donor excitation we use:

```
ds = d.select_bursts(select_bursts.size, th1=30)
```

The previous command creates a new *Data* variable (`ds`) containing the selected bursts. As mentioned before the new object will share the photon data arrays with the original object (`d`) in order to minimize the RAM use.

The first argument of `select_bursts` (link) is a python function implementing the "selection rule" (`select_bursts.size` in this example); all the remaining arguments (only `th1` in this case) are parameters of the selection rule. The `select_bursts` module (link) contains numerous built-in selection functions (link). For example, `select_bursts.ES` is used to select a region on the E-S ALEX histogram, `select_bursts.width` to select bursts based on their duration. New criteria can be easily implemented by defining a new selection function, which requires not more than a couple of lines of code in most cases (see the `select_bursts` module's source code for several examples, link).

Finally, different criteria can be combined by applying them sequentially. For example, with the following commands:

```
ds = d.select_bursts(select_bursts.size,
                     th1=50, th2=200)
dsw = ds.select_bursts(select_bursts.width,
                       th1=0.5e-3, th2=3e-3)
```

we apply a combined burst size and duration selection, in which bursts have sizes between 50 and 200 photons, and duration between 0.5 and 3 ms.

4.5.1 γ-corrected Burst Size Selection

In the previous section, we selected bursts by size using only photons detected by donor and acceptor channel during donor excitation. Conversely, we can apply a threshold on the all-photon burst size (section 3.5) by adding n_{aa} to the burst size as in eq. 2. This is achieved by passing `add_naa=True` to the selection function. When `add_naa` is not specified, as in the previous section, the default `add_naa=False` is used (i.e. use only photons during donor excitation). The complete selection command is:

```
ds = d.select_bursts(select_bursts.size,
                     th1=30, add_naa=True)
```

and the resulting selection is plotted in figure 4.

Another important parameter for defining the burst size is the γ-factor, i.e. the imbalance between the donor and the acceptor channels. As noted in section 3.5, the γ-factor is used to compensate bias for the different fluorescence quantum yields of the D and A fluorophores as well as the different photon-detection efficiencies of the D and A channels. When γ-factor is not 1, neglecting its effect on burst size leads to over-representing (in terms of number of bursts) one population versus to the others.

When the γ factor is known, users can pass the argument `gamma` during burst selection:

```
ds = d.select_bursts(select_bursts.size,
                     th1=15, gamma=0.65)
```

When not specified, $\gamma = 1$ is assumed.

For more information on burst size selection refer to the `select_bursts.size` documentation (link).

Python details The method `Data.burst_sizes` (link) computes and returns γ-corrected burst sizes with or without addition of `naa`.

4.5.2 Select the FRET Populations

In smFRET-ALEX experiments, in addition to one or more FRET populations, there are always donor-only (D-only) and acceptor-only (A-only) populations. In most cases, these additional populations are not of interest and need to be filtered out.

In principle, using the E-S representation, we can exclude D-only and A-only bursts by selecting bursts withing a range of S values (e.g. S=0.2-0.8). This approach, however, simply truncates the burst distribution with arbitrary thresholds and is therefore not recommended for quantitative assessment of FRET populations.

A better approach consists in applying two selection filters one after the other. First, we filter out the A-only population by applying a threshold on number of photons during donor excitation. Second, we exclude the D-only population by by applying a threshold on number of photons during acceptor excitation. The commands for this combined selections are:

```
ds1 = d.select_bursts(select_bursts.size, th1=15)
ds2 = ds1.select_bursts(select_bursts.naa, th1=15)
```

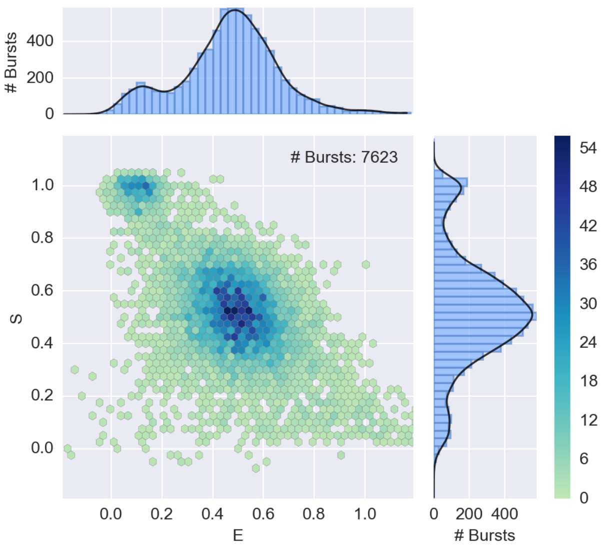

Figure 4: A 2-D ALEX histogram and marginal E and S histograms obtained with `alex_jointplot(ds)`. Bursts selected with a burst size threshold of 30 photons, including photons during the acceptor excitation. Three populations are visible: FRET population (middle), D-only population (top left) and A-only population (bottom, $S < 0.2$). Compare with figure 5 where the FRET population has been isolated.

Here, the variable `ds2` contains the composite selection of bursts. Figure 5 shows the resulting pure FRET population obtained with the previous selection.

4.6 Population Analysis

Typically, after bursts selection, E or S histograms are fitted to a model. FRETBursts `mfit` module allows fitting histograms of bursts quantities (i.e. E or S) with arbitrary models. In this context, a model is an object specifying a function, the parameters varied during the fit and optional constraints for these parameters. This concept of model is taken from *lmfit* [28], the underlying library used by FRETBursts to perform the fits.

Models can be created from arbitrary functions. For convenience, FRETBursts allows to use predefined models such as 1 to 3 Gaussian peaks or 2-Gaussian plus “bridge”. Built-in models are created calling a corresponding factory function (names starting with `mfit.factory_`) which initializes the parameters with values and constraints suitable for E and S histograms fits. (see *Factory Functions* documentation, link).

Continuing our example, in order to fit the E histogram of bursts in the `ds` variable with two Gaussian peaks, we use the following command:

```
bext.bursts_fitter(ds, 'E', binwidth=0.03,
                   model=mfit.
                       factory_two_gaussians())
```

Changing `'E'` with `'S'` will fit the S histogram instead. The argument `binwidth` specifies the histogram bin width and the argument `model` takes pre-initialized model used to be used for fitting.

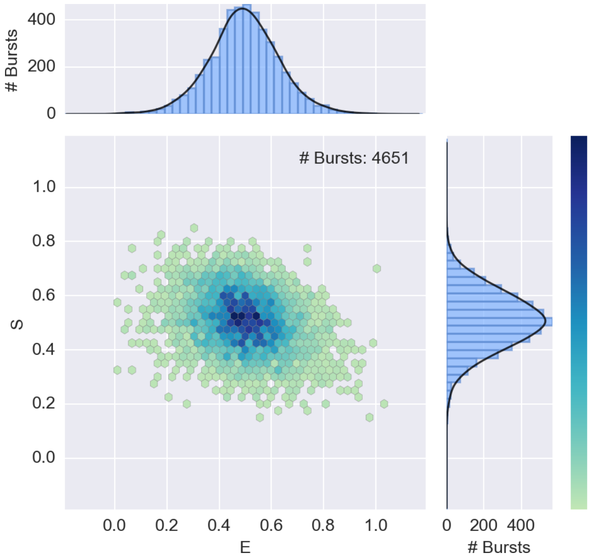

Figure 5: 2-D ALEX histogram after selection of FRET population using two selection filters on donor excitation and acceptor excitation number of photons, starting from bursts in figure 4.

All fitting results (including best fit values, uncertainties, etc...), are stored in the `E_fitter` (or `S_fitter`) attributes of the `Data` variable (here named `ds`). To print a comprehensive summary of the fitting results including uncertainties, reduced χ^2 and correlation between parameters we can use the following command:

```
fit_res = ds.E_fitter.fit_res[0]
print(fit_res.fit_report())
```

To plot the fitted model together with the FRET histogram as in figure 6, we pass the parameter `show_model=True` to `hist_fret` function as follows (see section 3.6 for an introduction to plotting):

```
dplot(ds, hist_fret, show_model=True)
```

For more examples on fitting bursts data and plotting results see the fitting section of the µs-ALEX notebook (link), the *Fitting Framework* section of the documentation (link) as well as the `bursts_fitter` function documentation (link).

Python details Models returned by FRETBursts’s factory functions (`mfit.factory_*`) are `lmfit.Model` objects (link). Custom models can be created calling `lmfit.Model` directly. When an `lmfit.Model` is fitted, it returns a `ModelResults` object (link) which contains all the information related to the fit (model, data, parameters with best values and uncertainties) and useful methods to operate on fit results. FRETBursts puts a `ModelResults` object of each excitation spot in the list `ds.E_fitter.fit_res`. As an example, to get the reduced χ^2 value of the E histogram fit in a single-spot measurement `d`, we use:

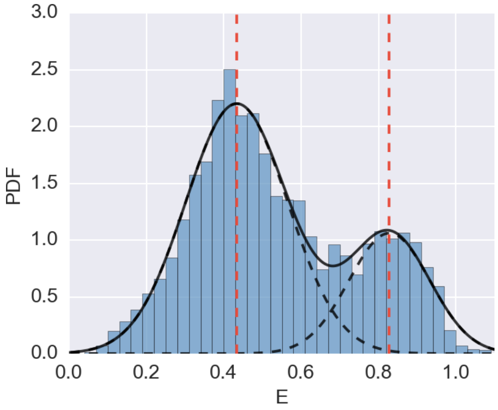

Figure 6: Example of a FRET histogram fitted with a 2-Gaussian model. The plot is generated after performing the fit with the command `dplot(ds, hist_fret, show_model=True)`.

```
d.E_fitter.fit_res[0].redchi
```

Other useful attributes are `aic` and `bic` which contain the Akaike information criterion (AIC) and the Bayes Information criterion (BIC) quantities. AIC and BIC allow to compare different models and to select the most appropriate for the data at hand.

Example of defining and modifying models for fitting are provided in the afore mentioned µs-ALEX notebook. Users can also refer to the comprehensive lmfit's documentation (link).

4.7 Timestamps and Burst Data

Beyond providing prepackaged functions for established methods, FRETBursts also provides the infrastructure for exploring new analysis approaches. Users can easily get timestamps (or selection masks) for any photon stream. Core burst data (essentially start and stop times, indexes and derived quantities for each burst) are stored in a `Bursts` object (link). This object provides a simple and well-tested interface (100 % unit-test coverage) to access and manipulate burst data. `Bursts` are created from a sequence of start/stop times and indexes, while all the other fields are automatically computed. `Bursts`'s methods allow to recompute indexes relative to a different photon selection or recompute start and stop times relative to a new timestamps array. Additional methods perform fusion of nearby bursts or combination of two set of bursts (time intersection or union). This functionality is used for example to implement the DCBS. In conclusion, `Bursts` efficiently implements all the common operations performed with burst data, providing and easy-to-use interface and well tested algorithms. Leveraging `Bursts` methods, users can implement new types of analysis without wasting time implementing (and debugging) standard manipulation routines. Examples of working directly with timestamps, masks (i.e. photon selections) and burst data are provided in one of the FRETBursts notebooks (link). Section 5 provides a complete example on using FRETBurts to implement custom burst analysis techniques.

Python details Timestamps are stored in the `Data` attribute `ph_times_m`, which is a list or arrays, one array per excitation spot. In single-spot measurements the full timestamps array is accessed as `Data.ph_times_m[0]`. To get timestamps of arbitrary photon streams, users can use the method `Data.get_ph_times` (link). Photon streams are selected from the full (all-photon) timestamps array using boolean masks. The masks can be obtained using the method `Data.get_ph_mask` (link). `Bursts` objects internally store the start and stop times and indexes in a numpy array. The other fields are computed on-the-fly using class properties, so they are always up to date even if start and stop are modified. Iteration over `Bursts` is relatively fast, with performances similar to iterating through numpy rows.

5 Implementing Burst Variance Analysis

In this section we describe how to implement the burst variance analysis (BVA) [39]. FRETBurts provides well-tested, general-purpose functions for timestamps and burst data manipulation and therefore simplifies implementing custom burst analysis algorithms such as BVA.

5.1 BVA Overview

Single-molecule FRET histograms show more information than just mean FRET efficiencies. While, in general, several peaks indicate the presence of multiple subpopulations, a single peak cannot be a priori associated with a single FRET efficiency, unless a detailed shot-noise analysis is carried out [1, 29].

The width of a FRET distribution has a typical lower boundary set by shot noise, which is caused by the statistics of discrete photon-detection events. FRET distributions broader than the shot noise limit, can be ascribed to a static mixture of species with slightly different FRET efficiencies, or to a specie undergoing dynamic transitions (e.g. interconversion between multiple states, diffusion in a continuum of conformations, binding-unbinding events, etc...). By simply looking at the FRET histogram, in cases when there is single peak broader than shot-noise, it is not possible to discriminate between the static and dynamic case. The BVA method has been developed to address this issue of detecting the presence of dynamics in FRET distributions [39], and has been successfully applied to identify biomolecular processes with dynamics on the millisecond time-scale [34, 39].

The basic idea behind BVA is to slice bursts in sub-bursts with a fixed number of photons n, and to compare the empirical variance of acceptor counts across all sub-bursts in a burst with the theoretical shot-noise limited variance, dictated by the Binomial distribution. An empirical variance of sub-bursts larger than the shot-noise limited value indicates the presence of dynamics. Naturally, since the estimation of the sub-bursts variance is affected by uncertainty, BVA analysis provides and indication of an higher or lower probability of observing dynamics.

In a FRET (sub-)population distribution originating from a single static FRET efficiency, the sub-bursts acceptor counts N_a can be modeled as a Binomial-distributed random variable $N_a \sim \text{Binom}\{n, E\}$, where n is the number of photons in each sub-burst and E is the estimated population FRET efficiency. Note that, without approximation, we can replace E with PR and use the uncorrected counts. This is possible because, regardless of the molecular FRET efficiency, the detected counts are partitioned between donor and acceptor channel according to a Binomila distribution whit a p parameter equal to PR. The only approximation done here and in the following paragraphs is neglecting the presence background (a reasonable approximation since the backgrounds counts are in general a very small fraction of the total counts). We refer the interested reader to [39] for further discussion.

If N_a follows a binomial distribution, the random variable $E = N_a/n$, has standard deviation reported in eq. 3.

$$\text{Std}(E) = \sqrt{\frac{E(1-E)}{n}} \tag{3}$$

5.2 BVA Implementation

BVA analysis consists of four steps: 1) slicing bursts into sub-bursts containing a constant number of consecutive photons, *n*, 2) computing FRET efficiencies of each sub-burst, 3) calculating the empirical standard deviation (s_E) of sub-burst FRET efficiencies over the whole burst, and 4) comparing s_E to the expected standard deviation of a shot-noise limited distribution (eq. 3).

If, as in figure 7, the observed FRET efficiency distribution originates from a static mixture of FRET efficiency sub-populations (of different non-interconverting molecules), s_E of each burst is only affected by shot noise and will follow the expected standard deviation curve based on eq. 3. Conversely, if the observed distribution originates from biomolecules of a single specie, which interconverts between different FRET sub-populations in (times comparable to diffusion time), as in figure 8, s_E of each burst will be larger than the expected shot-noise-limited standard deviation, hence it will be placed above the shot-noise standard deviation curve (right panel on figure 8). For implementation details please refer to the BVA notebook included with FRETBursts (link).

6 Development and Contributions

Errors are an inevitable reality in any reasonably complex software. It is therefore critical to implement countermeasures to minimize the probability of introducing bugs and their potential impact [32, 44]. We strive to follow modern best-practices in software development which are summarized below.

In FRETBursts, we highly value source code readability, a property which can reduce the number of bugs by facilitating understanding and verifying the code. For this purpose, FRETBursts code-base is well commented (more that 35% of source code), follows the PEP8 python code style rules (link), and has docstrings in napoleon format (link).

Reference documentation is built with Sphinx (sphinx-doc.org) and all the API documents are automatically generated from docstrings. On each commit, documentation is automatically built and deployed on ReadTheDocs.org.

Unit tests cover most of the core algorithms, ensuring consistency and minimizing the probability of introducing bugs. The TravisCI (link) continuous integration service, executes the full test suite on each commit, timely reporting errors. As a rule, whenever a bug is discovered, the fix also includes a new test to ensure that the same bug cannot happen in the future. In addition to the unit tests, we include a regression-test notebook (link) to easily compares numerical results between two versions of FRETBursts. Additionally, the tutorials themselves are executed before each release as an additional test layer to ensure that no errors or regressions are introduced.

FRETBursts is openly developed using the GitHub platform. Contributors of any level of expertize are welcome in the projects and publically acknowledged. Contributions can be as simple as pointing out deficiencies in the documentation but can also be bug reports or corrections to the documentation or code. Users willing to implement new features are encouraged to open an Issue on GitHub and to submit a Pull Request. The open source nature of FRETBursts guarantees that contributions will remain available to the entire single-molecule community.

7 Conclusions

FRETBursts provides an open source implementation of state-of-the-art smFRET burst analysis accessible to the whole single-molecule community. FRETBursts implements several novel concepts which can lead to significantly more accurate results in specific situations: time-dependent background estimation, background dependent burst search threshold, burst weighting, burst selection based on γ-corrected burst sizes.

More importantly, FRETBursts provides a library of well-tested routines for timestamps and burst manipulation, making it an ideal environment to quickly develop and compare novel analytical techniques.

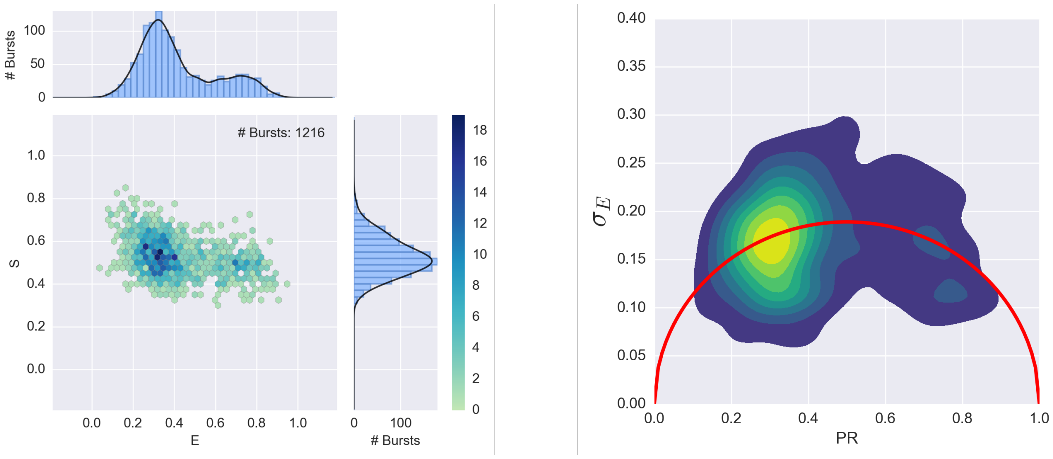

Figure 7: Left figure is an E-S histogram for a mixture of single stranded DNA (20dT) and double stranded DNA (20dT-20dA) in the presence of 200mM $MgCl_2$. Right figure is its BVA plot. Since both 20dT and 20dT-20dA are stable and have no dynamics, the BVA plots shows an s_E peak sitting on the expected standard deviation curve (*red curve*).

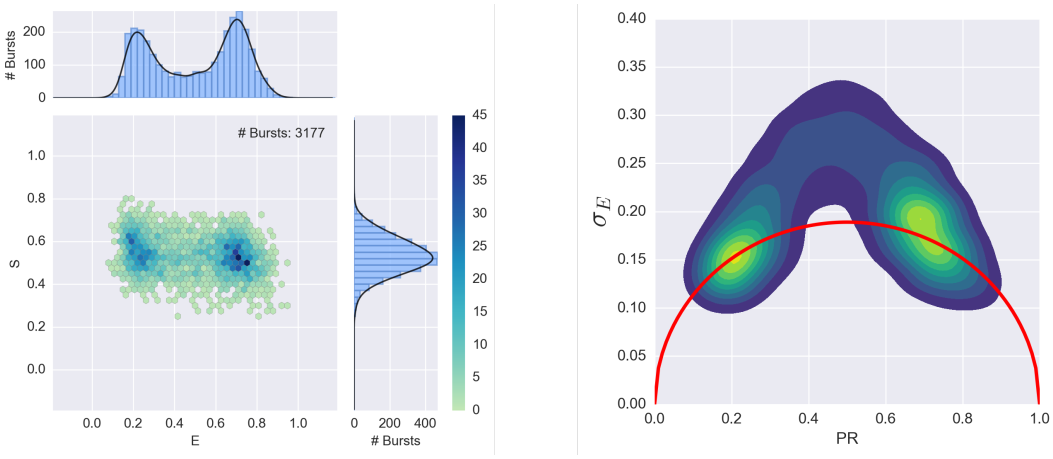

Figure 8: Left figure is an E-S histogram for single stranded DNA (A_{31}-TA, see in [40]), which is designed to form a hairpin structure temporarily and reversibly in the presence of 400mM NaCl. Right figure is its BVA plot. Since the transition between hairpin and open structure causes a significant change in FRET efficiency, the BVA plot shows that s_E, in the bridge region between two populations, sits largely above the expected standard deviation curve (*red curve*).

We summarize here what we consider to be the strengths of the FRETBursts software.

1. Open source and openly developed. The source code can be checked, modified and adapted for different purposes. All the software dependencies are open source as well.

2. Several state-of-the-art and novel algorithms for each step of the smFRET burst analysis pipeline are provided.

3. Modern software engineering design: the DRY principle to reduce duplication and the KISS principle to avoid over-engineering were followed.

4. Defensive programming [32]: care about code readability, unit and regression testing and continuous integration.

Given its features, FRETBursts is both a toolkit for developing novel algorithms in smFRET burst analysis and a well-tested software for standard smFRET burst analysis. Its open source nature and open development process assures that discovered bug are always timely disclosed, and lowers the barriers for new users to identify and potentially fix bugs, and even develop new or improved analysis.

Acknowledgments

This work was supported in part by National Institutes of Health (NIH) grant R01-GM95904 and by U.S. Department Energy (DOE) grant DEFC02-02ER63421-00.

References

[1] M. Antonik, S. Felekyan, A. Gaiduk, and C. A. M. Seidel. Separating structural heterogeneities from stochastic variations in fluorescence resonance energy transfer distributions via photon distribution analysis. *Journal of Physical Chemistry B*, 110(13):6970–6978, apr 2006.

[2] J. D. Blischak, E. R. Davenport, and G. Wilson. A Quick Introduction to Version Control with Git and GitHub. *PLOS Computational Biology*, 12(1):e1004668, jan 2016.

[3] J. B. Buckheit and D. L. Donoho. WaveLab and Reproducible Research. In *Wavelets and Statistics*, pages 55–81. Springer Science + Business Media, 1995.

[4] B. A. Camley, F. L. H. Brown, and E. A. Lipman. Förster transfer outside the weak-excitation limit. *J. Chem. Phys.*, 131(10):104509, 2009.

[5] M. Dahan, A. A. Deniz, T. Ha, D. S. Chemla, P. G. Schultz, and S. Weiss. Ratiometric measurement and identification of single diffusing molecules. *Chemical Physics*, 247(1):85–106, aug 1999.

[6] M. Droettboom, J. Hunter, T. A. Caswell, E. Firing, J. H. Nielsen, P. Elson, B. Root, D. Dale, J.-J. Lee, J. K. Seppänen, D. McDougall, A. Straw, R. May, N. Varoquaux, T. S. Yu, E. Ma, C. Moad, S. Silvester, C. Gohlke, P. Würtz, T. Hisch, F. Ariza, Cimarron, I. Thomas, J. Evans, P. Ivanov, J. Whitaker, P. Hobson, mdehoon, and M. Giuca. matplotlib: matplotlib v1.5.1, jan 2016.

[7] C. Eggeling, S. Berger, L. Brand, J. Fries, J. Schaffer, A. Volkmer, and C. Seidel. Data registration and selective single-molecule analysis using multiparameter fluorescence detection. *Journal of Biotechnology*, 86(3):163–180, apr 2001.

[8] C. Eggeling, J. R. Fries, L. Brand, R. Gunther, and C. A. M. Seidel. Monitoring conformational dynamics of a single molecule by selective fluorescence spectroscopy. *Proceedings of the National Academy of Sciences*, 95(4):1556–1561, feb 1998.

[9] J. Freeman. Open source tools for large-scale neuroscience. *Current Opinion in Neurobiology*, 32:156–163, jun 2015.

[10] J. R. Fries, L. Brand, C. Eggeling, M. Köllner, and C. A. M. Seidel. Quantitative Identification of Different Single Molecules by Selective Time-Resolved Confocal Fluorescence Spectroscopy. *J. Phys. Chem. A*, 102(33):6601–6613, aug 1998.

[11] I. Gopich and A. Szabo. Theory of photon statistics in single-molecule Förster resonance energy transfer. *J. Chem. Phys.*, 122(1):014707, 2005.

[12] I. V. Gopich. Concentration Effects in "Single-Molecule" Spectroscopy †. *J. Phys. Chem. B*, 112(19):6214–6220, may 2008.

[13] I. V. Gopich and A. Szabo. Single-Molecule FRET with Diffusion and Conformational Dynamics. *J. Phys. Chem. B*, 111(44):12925–12932, nov 2007.

[14] E. C. Hayden. Mozilla plan seeks to debug scientific code. *Nature*, 501(7468):472–472, sep 2013.

[15] E. C. Hayden. Rule rewrite aims to clean up scientific software. *Nature*, 520(7547):276–277, apr 2015.

[16] J. Hohlbein, T. D. Craggs, and T. Cordes. Alternating-laser excitation: single-molecule FRET and beyond. *Chem. Soc. Rev.*, 43(4):1156–1171, 2014.

[17] D. C. Ince, L. Hatton, and J. Graham-Cumming. The case for open computer programs. *Nature*, 482(7386):485–488, feb 2012.

[18] A. Ingargiola, T. Laurence, R. Boutelle, S. Weiss, and X. Michalet. Photon-HDF5: An Open File Format for Timestamp-Based Single-Molecule Fluorescence Experiments. *Biophysical Journal*, 110(1):26–33, jan 2016.

[19] A. Ingargiola, F. Panzeri, N. Sarkhosh, A. Gulinatti, I. Rech, M. Ghioni, S. Weiss, and X. Michalet. 8-spot smFRET analysis using two 8-pixel SPAD arrays. In J. Enderlein, I. Gregor, Z. K. Gryczynski, R. Erdmann, and F. Koberling, editors, *Single Molecule Spectroscopy and Superresolution Imaging VI*. SPIE, feb 2013.

[20] A. N. Kapanidis, T. A. Laurence, N. K. Lee, E. Margeat, X. Kong, and S. Weiss. Alternating-Laser Excitation of Single Molecules. *Acc. Chem. Res.*, 38(7):523–533, jul 2005.

[21] A. N. Kapanidis, E. Margeat, S. O. Ho, E. Kortkhonjia, S. Weiss, and R. H. Ebright. Initial Transcription by RNA Polymerase Proceeds Through a DNA-Scrunching Mechanism. *Science*, 314(5802):1144–1147, nov 2006.

[22] T. A. Laurence, X. Kong, M. Jäger, and S. Weiss. Probing structural heterogeneities and fluctuations of nucleic acids and denatured proteins. *PNAS*, 102(48):17348–17353, 2005.

[23] N. K. Lee, A. N. Kapanidis, Y. Wang, X. Michalet, J. Mukhopadhyay, R. H. Ebright, and S. Weiss. Accurate FRET Measurements within Single Diffusing Biomolecules Using Alternating-Laser Excitation. *Biophysical Journal*, 88(4):2939–2953, apr 2005.

[24] E. Lerner, T. Orevi, E. Ben Ishay, D. Amir, and E. Haas. Kinetics of fast changing intramolecular distance distributions obtained by combined analysis of FRET efficiency kinetics and time-resolved FRET equilibrium measurements. *Biophysical journal*, 106(3):667–76, feb 2014.

[25] X. Michalet, R. A. Colyer, G. Scalia, A. Ingargiola, R. Lin, J. E. Millaud, S. Weiss, O. H. W. Siegmund, A. S. Tremsin, J. V. Vallerga, A. Cheng, M. Levi, D. Aharoni, K. Arisaka, F. Villa, F. Guerrieri, F. Panzeri, I. Rech, A. Gulinatti, F. Zappa, M. Ghioni, and S. Cova. Development of new photon-counting detectors for single-molecule fluorescence microscopy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 368(1611):20120035–20120035, dec 2012.

[26] R. R. Murphy, S. E. Jackson, and D. Klenerman. pyFRET: A Python Library for Single Molecule Fluorescence Data Analysis. *ArXiv*, dec 2014.

[27] B. K. Müller, E. Zaychikov, C. Bräuchle, and D. C. Lamb. Pulsed Interleaved Excitation. *Biophysical Journal*, 89(5):3508–3522, nov 2005.

[28] M. Newville, T. Stensitzki, D. B. Allen, and A. Ingargiola. LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python¶, sep 2014.

[29] E. Nir, X. Michalet, K. M. Hamadani, T. A. Laurence, D. Neuhauser, Y. Kovchegov, and S. Weiss. Shot-Noise Limited Single-Molecule FRET Histograms: Comparison between Theory and Experiments †. *J. Phys. Chem. B*, 110(44):22103–22124, nov 2006.

[30] W. C. Parr and W. R. Schucany. Minimum Distance and Robust Estimation. *Journal of the American Statistical Association*, 75(371):616, sep 1980.

[31] C. Pradal, G. Varoquaux, and H. P. Langtangen. Publishing scientific software matters. *Journal of Computational Science*, 4(5):311–312, sep 2013.

[32] A. Prlić and J. B. Procter. Ten Simple Rules for the Open Development of Scientific Software. *PLoS Computational Biology*, 8(12):e1002802, dec 2012.

[33] G. Rahamim, M. Chemerovski-Glikman, S. Rahimipour, D. Amir, and E. Haas. Resolution of Two Sub-Populations of Conformers and Their Individual Dynamics by Time Resolved Ensemble Level FRET Measurements. *PloS one*, 10(12):e0143732, jan 2015.

[34] N. C. Robb, T. Cordes, L. C. Hwang, K. Gryte, D. Duchi, T. D. Craggs, Y. Santoso, S. Weiss, R. H. Ebright, and A. N. Kapanidis. The Transcription Bubble of the RNA Polymerase–Promoter Open Complex Exhibits Conformational Heterogeneity and Millisecond-Scale Dynamics: Implications for Transcription Start-Site Selection. *Journal of Molecular Biology*, 425(5):875–885, mar 2013.

[35] Y. Santoso, J. P. Torella, and A. N. Kapanidis. Characterizing Single-Molecule FRET Dynamics with Probability Distribution Analysis. *ChemPhysChem*, 11(10):2209–2219, jun 2010.

[36] H. Shen. Interactive notebooks: Sharing the code. *Nature*, 515(7525):151–152, nov 2014.

[37] M. A. Stephens. EDF Statistics for Goodness of Fit and Some Comparisons. *Journal of the American Statistical Association*, 69(347):730, sep 1974.

[38] T. E. Tomov, R. Tsukanov, R. Masoud, M. Liber, N. Plavner, and E. Nir. Disentangling Subpopulations in Single-Molecule FRET and ALEX Experiments with Photon Distribution Analysis. *Biophysical Journal*, 102(5):1163–1173, mar 2012.

[39] J. P. Torella, S. J. Holden, Y. Santoso, J. Hohlbein, and A. N. Kapanidis. Identifying Molecular Dynamics in Single-Molecule FRET Experiments with Burst Variance Analysis. *Biophysical Journal*, 100(6):1568–1577, mar 2011.

[40] R. Tsukanov, T. E. Tomov, R. Masoud, H. Drory, N. Plavner, M. Liber, and E. Nir. Detailed Study of DNA Hairpin Dynamics Using Single-Molecule Fluorescence Assisted by DNA Origami. *The Journal of Physical Chemistry B*, 117(40):11932–11942, oct 2013.

[41] M. Vihinen. No more hidden solutions in bioinformatics. *Nature*, 521(7552):261–261, may 2015.

[42] M. Waskom, O. Botvinnik, P. Hobson, J. Warmenhoven, J. B. Cole, Y. Halchenko, J. Vanderplas, S. Hoyer, S. Villalba, E. Quintero, A. Miles, T. Augspurger, T. Yarkoni, C. Evans, D. Wehner, L. Rocher, T. Megies, L. P. Coelho, E. Ziegler, T. Hoppe, S. Seabold, S. Pascual, P. Cloud, M. Koskinen, C. Hausler, kjemmett, D. Milajevs, A. Qalieh, D. Allan, and K. Meyer. seaborn: v0.6.0 (June 2015), jun 2015.

[43] S. Weiss. Fluorescence Spectroscopy of Single Biomolecules. *Science*, 283(5408):1676–1683, mar 1999.

[44] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson. Best Practices for Scientific Computing. *PLoS Computational Biology*, 12(1):e1001745, 01 2014.

[45] K. Zhang and H. Yang. Photon-by-Photon Determination of Emission Bursts from Diffusing Single Chromophores. *J. Phys. Chem. B*, 109(46):21930–21937, nov 2005.