FRETBursts: An Open Source Toolkit for Analysis of Freely-Diffusing Single-Molecule FRET

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

1 Dept. Chemistry and Biochemistry, Univ. of California Los Angeles, Los Angeles, CA, USA

* ingargiola.antonino@gmail.com

Abstract

Single-molecule Förster Resonance Energy Transfer (smFRET) allows probing intermolecular interactions and conformational changes in biomacromolecules, and represents an invaluable tool for studying cellular processes at the molecular scale. 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, which can be detected by single-photon avalanche diode (SPAD) detectors. The intensities of donor and acceptor fluorescence can then be related to the distance between the two fluorophores.

While in recent years we observed a growing number of contributions proposing improvements or new techniques in smFRET data analysis, rarely those publications were accompanied by a software implementation. Remarkably, given the widespread application of smFRET, no complete software package for smFRET burst analysis is freely available to date.

In this paper, we introduce FRETBursts, an open source software for analysis of freely-diffusing smFRET data. FRETBursts allows executing all the fundamental steps of smFRET bursts analysis using state-of-the-art as well as novel techniques, while providing an open, robust and well-documented implementation. Therefore, FRETBursts represents an ideal platform for comparison and development of new methods in burst analysis.

We employ modern software engineering principles in order to minimize bugs and facilitate long-term maintainability. Furthermore, we place a strong focus on reproducibility by relying on Jupyter notebooks for FRETBursts execution. Notebooks are executable documents capturing all the steps of the analysis (including data files, input parameters, and results) and can be easily shared to replicate complete smFRET analyzes. Notebooks allow beginners to execute complex workflows and advanced users to customize the analysis for their own needs. By bundling in a single document analysis description, code and results, FRETBursts allows to seamless share analysis workflows and results encourages reproducibility and facilitates collaboration among researchers in the single-molecule community.

1 Introduction

1.1 Open Science and Reproducibility

Over the past 20 years, single molecule FRET (smFRET) has grown into one of the most useful techniques in single-molecule spectroscopy [1, 2]. While it is possible to extract information on sub-populations using ensemble measurements (e.g. [3, 4]), smFRET unique feature is its ability to very straightforwardly resolve conformational changes of biomolecules or measure binding-unbinding kinetics in heterogeneous samples [5–9]. smFRET measurements on freely diffusing molecules (the focus of this paper) have the additional advantage over measurements performed on immobilized molecules, of allowing to probe molecules and processes without perturbation from surface immobilization or additional functionalization needed for surface attachment [10, 11].

The increasing amount of work using freely-diffusing smFRET has motivated a growing number of theoretical contributions to the specific topic of data analysis [12–24]. Despite this profusion of publications, most research groups still rely on their own implementation of a limited number of methods, with very little collaboration or code sharing. To clarify this statement, let us point that our own group's past smFRET papers merely mention the use of custom-made software without additional details [16, 17]. Even though some of these software tools are made available upon request, or sometimes shared publicly on websites, it remains hard to reproduce and validate results from different groups, let alone build upon them. Additionally, as new methods are proposed in literature, it is generally difficult to quantify their performance compared to other methods. An independent quantitative assessment would require a complete reimplementation, an effort few groups can afford. As a result, potentially useful analysis improvements are either rarely or slowly adopted by the community. In contrast with other established traditions such as sharing protocols and samples, in the domain of scientific software, we have relegated ourselves to islands of non-communication.

From a more general standpoint, the non-availability of the code used to produce scientific results, hinders reproducibility, makes it impossible to review and validate the software's correctness and prevents improvements and extensions by other scientists. This situation, common in many disciplines, represents a real impediment to the scientific progress. Since pioneering work of Donoho group in the 90's [25], it has become evident that developing and maintaining open source scientific software for reproducible research is a critical requirement of the modern scientific enterprise [26, 27].

Other disciplines have started tackling this issue [28], and even in the single-molecule field a few recent publications have provided software for analysis of surface-immobilized experiments [29–33]. For freely-diffusing smFRET experiments, although it is common to find mention of "code available from the authors upon request" in publications, there is a dearth of such open source code, with, to our knowledge, the notable exception of a single example [34]. To address this issue, we have developed FRETBursts, an open source Python software for analysis of freely-diffusing single-molecule FRET measurements. FRETBursts can be used, inspected and modified by anyone interested in using state-of-the art smFRET analysis methods or implementing modifications or completely new techniques. FRETBursts therefore represents an ideal platform for quantitative comparison of different methods for smFRET burst analysis. Technically, a strong emphasis has been given to the reproducibility of complete analysis workflows. FRETBursts uses Jupyter Notebooks [35], an interactive and executable document containing textual narrative, input parameters, code, and computational results (tables, plots, etc.). A notebook thus captures the various analysis steps in a document which is easy to share and execute. To minimize the possibility of bugs being introduced

inadvertently [36] we employ modern software engineering techniques such as unit testing and continuous integration [28, 37]. FRETBursts is hosted on GitHub [38, 39], where users can write comments, report issues or contribute code. In a related effort, we recently introduced Photon-HDF5 [40], an open file format for timestamp-based single-molecule fluorescence experiments. An other related open source tool is PyBroMo [41], a freely-diffusing smFRET simulator which produces Photon-HDF5 files that are directly analyzable with FRETBursts. Together with all the aforementioned tools, FRETBursts contributes to the growing ecosystem of open tools for reproducible science in the single-molecule field.

1.2 Paper Overview

This paper is written as an introduction to smFRET burst analysis and its implementation in FRETBursts. After a brief overview of FRETBursts features (section 2), we introduce essential concepts and terminology for smFRET burst analysis (section 3).

In section 4, we illustrate the steps involved in smFRET burst analysis: (i) data loading (section 4.1), (ii) definition of the excitation alternation periods (section 4.2), (iii) background correction (section 4.3), (iv) burst search (section 4.4), (v) burst selection (section 4.5) and (vi) FRET histogram fitting (section 4.6). The aim of this section is to illustrate the specificities and trade-off involved in various approaches with sufficient details to enable readers to customize the analysis for their own needs. Section 5 walks the reader thorough implementing Burst Variance Analysis (BVA) [23], as an example of implementation of an advanced data processing technique. Finally, section 6 summarizes 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 more legible, we have concentrated Python-specific details in subsections entitled *Python details*. These subsections provide deeper insights for readers already familiar with Python and can be initially skipped by readers who are not. Finally, note that all commands and figures in this paper can be regenerated using the accompanying notebooks (link).

2 FRETBursts Overview

2.1 Technical Features

FRETBursts can analyze smFRET measurements from one or multiple excitation spots [42]. The supported excitation schemes include single laser, alternating laser excitation (ALEX) with either CW lasers (µs-ALEX [43]) or pulsed lasers (ns-ALEX [44] or pulsed-interleaved excitation (PIE) [45]).

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 [10], dual-channel burst search (DCBS) [17] 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 signal-to-background ratio [46]). Moreover, FRETBursts provides a large set of fitting options to characterize FRET subpopulations. 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

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)

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.

Additionally, implementations of population dynamics analysis such as Burst Variance Analysis (BVA) [23] and two-channel kernel density distribution estimator (2CDE) [24] are available as FRETBursts notebooks.

2.2 Software Availability

FRETBursts is hosted and openly developed on GitHub. FRETBursts homepage (link) contains links to the various resources. Installation instructions can be found in the Reference Documentation (link). A description of FRETBursts execution using Jupyter notebooks is reported in SI 7.1. Detailed information on development style, testing strategies and contributions guidelines are reported in SI 7.2. Finally, to facilitate evaluation and comparison with other software, we setup an on-line services allowing to execute FRETBursts without requiring any installation on the user's computer (link).

3 Architecture and Concepts

In this section, we introduce some general concepts and notations used in FRETBursts.

3.1 Photon Streams

The raw data collected during a smFRET experiment consists in one or more arrays of photon timestamps, whose temporal resolution is set by the acquisition hardware, typically between 10 and 50 ns. In single-spot measurements, all timestamps are stored in a single array. In multispot measurements [42], there are as many timestamps arrays as excitation spots.

Each array contains timestamps from both donor (D) and acceptor (A) channels. When alternating excitation lasers are used (ALEX measurements) [16], a further distinction between photons emitted during the D or A excitation periods can be made. In FRETBursts, the corresponding sets of photons are called "photon streams" and are specified with a `Ph_sel` object (link). In non-ALEX smFRET data, there are 3 photon streams (table 1), while in ALEX data, there are 5 streams (table 2).

The `Ph_sel` class (link) allows the specification of any combination of photon streams. For example, in ALEX measurements, the D-emission during A-excitation stream is usually ignored because it does not contain any useful signal [16]. To indicate all but photons in this photon stream, the syntax is `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 to both select a proper threshold for burst search, and correct the raw burst counts by subtraction of background counts.

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 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 characterized by a lower count rate, due to "background counts" originating from detector dark counts, afterpulsing, out-of-focus molecules and sample scattering and/or impurities [20, 47]. The signature of these two types of processes can be observed in the inter-photon delays distribution (i.e. the waiting times between two subsequent timestamps) as illustrated in figure 1(a). The "tail" of the distribution (a straight line in semi-log scale) corresponds to exponentially-distributed time-delays, indicating that those counts are generated by a Poisson process. At short timescales, the distribution departs from the exponential due to the contribution of the higher rate process of single molecules traversing the excitation volume. To estimate the background rate (i.e. the inverse of the exponential time constant), it is necessary to define a time-delay threshold above which the distribution can be considered exponential. Finally, a parameter estimation method needs to be specified, such as Maximum Likelihood Estimation (MLE) or non-linear least squares curve fitting of the time-delay histogram (both supported in FRETBursts).

Figure 1. Inter-photon delays fitted with and exponential function. 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 background rate.

It is advisable to monitor the background as a function of time throughout the measurement, in order to account for possible variations. Experimentally, we found that when the background is not constant, it usually varies on time scales of tens of seconds (see figure 2). FRETBursts divides the acquisition in constant-duration time windows called *background periods* and computes the background rates for each of these windows (see section 4.3). Note that FRETBursts uses these local background rates also during burst search, in order to compute time-dependent burst detection thresholds and for background correction of burst data (see section 4.4).

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 contain the number of

Figure 2. Background rates as a function of time. Estimated background rate as a function of time for two µs-ALEXmeasurements. 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 b*) A measurement performed on an unsealed sample exhibiting significant background variations due to sample evaporation and/or photo-bleaching (likely of impurities the cover-glass). 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.

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.

photons detected in the donor or the 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 lists of arrays (or scalars) with the length of the lists equal to the number of excitation spots. This means that in single-spot measurements, an array of burst-data is accessed by specifying the index as 0, for example `Data.nd[0]`. `Data` implements a shortcut syntax to access the first element of a list with an underscore, so that an equivalently syntax is `Data.nd_` instead of `Data.nd[0]`.

3.4 Introduction to Burst Search

Identifying single-molecule fluorescence 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 ([10], [12]), 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 ensures that all resulting bursts have a signal-to-background ratio (SBR) larger than $(F-1)$ [46]. 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 sub-populations, the burst search should use all photons (i.e. APBS). 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 performing the burst search on arbitrary selections of photons. (see section 3.1 for more information on photon stream definitions).

Additionally, Nir *et al.* [17] proposed DCBS ('dual-channel burst search'), which can help mitigating artifacts due to photophysics effects such as blinking. During DCBS, a search is performed in parallel on two photon streams and bursts are defined as periods during which both photon streams exhibit a rate higher than the threshold, implementing the equivalent of an AND logic operation. Conventionally, 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')`). In FRETBursts, the user can choose arbitrary photon streams as input, an in general this kind of search is called a 'AND-gate burst search'.

After burst search, it is necessary to select bursts, for instance by specifying a minimum number of photons (or 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.* [10]. This method, however, neglects the effect of background and γ factor on the burst size and can lead to a selection bias for some channels and/or sub-populations. For this reason, we suggest performing a burst size selection after background correction, taking into account the γ factor, as discussed in sections 3.5 and 4.5. In special cases, users may choose to replace (or combine) the burst selection based on burst size with another criterion such as burst duration or brightness (see section 4.5).

3.5 Corrected Burst Sizes and Weights

The number of photons detected during a burst –the "burst size"– is 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. The factor γ takes into account different fluorescence quantum yields of donor and acceptor fluorophores and different photon detection efficiencies between donor and acceptor detection channels [16, 48]. Eq. 1 includes counts collected during donor excitation periods only, while eq. 2 includes all counts. 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 follow an exponential distribution. Compared to bursts with smaller sizes, bursts with large sizes are less frequent, but contain more information per-burst (having higher SNR). Therefore, selecting bursts by size is an important step (see section 4.5). A threshold set too low may result in unresolvable sub-populations because of broadening of FRET peaks and appearance of shot-noise artifacts in the FRET (and S) distribution (i.e. spurious narrow peaks due to E and S being computed as the ratio of small integers).

Conversely, too large a threshold may result in too low a number of bursts therefore poor representation of the FRET distribution. Additionally, especially when computing fractions of sub-populations (e.g. ratio of number of bursts in each sub-population), it is important to use γ-corrected burst sizes as selection criterion, in order to avoid under-representing some FRET sub-populations due to different quantum yields of donor and acceptor dyes and/or different photon detection efficiencies of donor and acceptor channels.

A simple way to mitigate the dependence of the FRET distribution on the burst size selection threshold is weighting bursts proportionally to their size so that the bursts with largest sizes will have the largest weights. Using size as weights (instead of any other monotonically increasing function of size) can be justified noticing that the variance of bursts PR (E_i) is inversely proportional to the burst size (see SI 7.6 for details).

In general, a weighting scheme is used for building efficient estimators for a population parameter (e.g. E_p). But, it can also be used to build weighted histograms or Kernel Density Estimation (KDE) plots which emphasize FRET subpopulations peaks without excluding small size bursts. Traditionally, for optimal results when not using weights, the FRET histogram is manually adjusted by finding an ad-hoc (high) size-threshold which selects only bursts with the highest size (and thus lowest variance). Building size-weighted FRET histograms is a simple method to balance the need of reducing the peaks width with the need of including as much bursts as possible to reduce statistical noise. As a practical example, by fixing the burst size threshold to a low value (e.g. 10-20 photons) and using weights, is possible to build a FRET histogram with well-defined FRET sub-populations peaks without the need of searching an optimal burst-size threshold (SI 7.6).

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). Other weighting functions (for example depending quadratically on the size) are listed in the `fret_fit.get_weights` documentation (link). However, using weights different from the size is not recommended due to their less efficient use of burst information.

4 smFRET Burst Analysis

4.1 Loading the Data

While FRETBursts can load several data files formats, we encourage users to adopt the recently introduced Photon-HDF5 file format [40]. Photon-HDF5 is 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 supports compression and allows saving photon data (e.g. timestamps) and measurement-specific metadata (e.g. setup and sample information, authors, provenance, etc.). Moreover, Photon-HDF5 is designed for long-term data preservation and aims to facilitate data sharing between different software and research groups. All example data files provided with FRETBursts use the Photon-HDF5 format.

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

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

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

The same command can load data from a variety of smFRET measurements supported by the Photon-HDF5 format, taking advantage of the rich metadata included with each file. For instance, data generated using different excitation schemes such as CW excitation or pulsed excitation, single-laser vs two alternating lasers, etc., or with any number of excitation spots, are automatically recognized and interpreted accordingly.

FRETBursts also supports loading µs-ALEXdata stored in .sm files (a custom binary format used in the Weiss lab), ns-ALEX data stored in .spc files (a binary format used by TCSPC Becker & Hickl acquisition hardware). Alternatively, these and other formats (such as ht3, a binary format used by PicoQuant hardware) can be converted into Photon-HDF5 files using phconvert, a file conversion library and utility for Photon-HDF5 (link). More information on loading different file formats can be found in the `loader` module's documentation (link).

4.2 Alternation Parameters

For µs-ALEXand ns-ALEX data, Photon-HDF5 normally stores parameters defining alternation periods corresponding to donor and acceptor laser excitation. At load time, a user can plot these parameters and change them if deemed necessary. In µs-ALEXmeasurements [49], CW laser lines are alternated on timescales of the order of 10 to 100 µs. Plotting an histogram of timestamps modulo the alternation period, it is possible to identify the donor and acceptor excitation periods (see figure 3a). In ns-ALEX measurements [44], pulsed lasers with equal repetition rates are delayed with respect to one another with typical delays of 10 to 100 ns. In this case, forming an histogram of TCSPC times (nanotimes) will allow the definition of periods of fluorescence after excitation of either the donor or the acceptor (see figure 3b). In both cases, the function `plot_alternation_hist` (link) will plot the relevant alternation histogram (figure 3) using currently selected (or default) values for donor and acceptor excitation periods.

Figure 3. Alternation histograms for µs-ALEXand ns-ALEX measurements. Histograms used for the selection/determination of the alternation periods for two typical smFRET-ALEX experiments. Distributions of photons detected by donor channel are in *green*, and by acceptor channel in *red*. The light *green* and *red* shaded areas indicate the donor and acceptor period definitions. (a) µs-ALEXalternation histogram, i.e. histogram of timestamps *modulo* the alternation period for a smFRET measurement. (b) ns-ALEX nanotime histogram for a smFRET measurement. 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 (link).

To change the period definitions, we can type:

```
d.add(D_ON=(2100, 3900), A_ON=(100, 1900))
```

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 for both µs-ALEXand ns-ALEX measurements. After changing the parameters, a new alternation plot will show the updated period definitions.

The alternation period definition can be applied to the data 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. If the user needs to update the periods definition, the data file will need to be reloaded and the steps above repeated as described.

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 fixed threshold of 2 ms for the all photon streams, the corresponding command is:

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

The first argument (`bg.exp_fit`) is the function used to fit the background rate 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. The second argument, `time_s`, is the duration of the *background period* (section 3.2) and the third, `tail_min_us`, is the minimum inter-photon delay to use when fitting the distribution to the specified model function. To use different thresholds for each photon stream we pass a tuple (i.e. a comma-separated list of values, link) instead of a scalar. The recommended approach is however automating the choice of threshold using `tail_min_us='auto'` using an heuristic algorithm which is described in *Background estimation* section of the µs-ALEXtutorial (link). Finally, it is possible to use a slower but rigorous approach for finding the optimal threshold as described in SI 7.5.

FRETBursts provides two kinds of plots to represent the background. One shows the histograms of inter-photon delays compared to the fitted exponential distribution, shown in figure 1) (see section 3.2 for details on the inter-photon distribution). This plot is created with the command:

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

This command reflects the general form of plotting commands in FRETBursts as described in SI 7.4. Here we only note that the argument `period` is an integer specifying the background period to be plotted (When omitted, the default is 0, i.e. the first period). Figure 1 allows to quickly identify pathological cases where the background fitting procedure returns unreasonable values.

The second background-related plot represents a timetrace of background rates, as shown in figure 2. This plot allows monitoring background rate variations occurring during the measurement and is obtained with the command:

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

Normally, samples should have a fairly constant background rate as a function of time as in figure 2(a). However, sometimes, non-ideal experimental conditions can yield a time-varying background rate, as illustrated in figure 2(b). A possible reason for the observed behavior could be buffer evaporation from an open sample or poorly sealed observation chamber. Alternatively, cover-glass impurities can contribute to the background. These impurities tend to bleach on timescales of minutes resulting in background variations during the course of the measurement.

Python details The estimated background rates are stored in the `Data` attributes `bg_dd`, `bg_ad` and `bg_aa`, corresponding to 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 time windows (background periods). Additional background fitting functions (e.g. least-square fitting of inter-photon delay histogram) are available in `bg` namespace (i.e. the `background` module, link).

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 using 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 count rate threshold equal to 6 times the local background rate (`F=6`), using 10 consecutive photons to compute the local count rate (`m=10`). A different photon stream, threshold (F) or number of photons m can be selected by passing different values. These parameters are good general-purpose starting point for smFRET analysis but can they can be adjusted if needed.

Note that the previous burst search does not perform any burst size selection (however, by definition, the minimum bursts size is effectively m). An additional parameter L can be passed to impose a minimum burst size before any correction. However, it is recommended to select bursts only after background corrections are applied, as discussed 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 counts-per-second). For example, a burst search with a 50 kcps threshold is 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 illustrated 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 of the `Data` variable `d` (in this example, the copy is called `d_dcbs`). Since FRETBursts shares the timestamps and detectors arrays between different copies of `Data` objects, the memory usage is minimized, even when several copies are created.

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 those used 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 leakage into the acceptor channel and acceptor direct excitation by the donor excitation laser [16]. These corrections can be applied to burst data by simply assigning values to the respective `Data` attributes:

```
d.gamma = 0.85
d.leakage = 0.15
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, when the goal is to fit the FRET efficiency of sub-populations, it is simpler to fit the uncorrected FRET histogram (i.e. background-corrected proximity ratio) and then correct the fitted FRET efficiency (see SI in [16], SI). Correcting the PR after fitting (instead of correcting the data in each burst) avoids distortion of the FRET distribution and keeps peaks of static FRET subpopulations closer to the ideal Binomial statistics [19].

FRETBursts implements the correction formulas for E and S in the functions `fretmath.correct_E_gamma_leak_dir` and `fretmath.correct_S` (link). A derivation of these correction formulas (using computer-assisted algebra) can be found online as an interactive 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 burst size.

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

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

The previous command creates a new `Data` variable (`ds`) containing the selected bursts. `th1` defines the lower bound for burst size, while `th2` defines the upper bound (when not specified, as in the previous example, the upper bound is $+\infty$). As before, the new object will share the photon data arrays with the original object (`d`) in order to minimize the amount of used memory.

The first argument of `select_bursts` (link) is a python function implementing the "selection rule" (`select_bursts.size` in this example); all 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 custom criteria can be readily implemented by defining a new selection function, which requires only a couple of lines of code (see the `select_bursts` module's source code for examples, link).

Finally, different criteria can be combined sequentially. For example, 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)
```

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

4.5.1 Burst Size Selection

In the previous section, we selected bursts by size, using only Dex photons (i.e. photons detected in both D and A channels during D excitation, as in eq. 1). Alternatively, a threshold on the burst size computed including all photons can be applied by adding n_{aa} to the burst size (see 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. computed size using only Dex photons). The complete selection command is:

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

The result of this selection is plotted in figure 4.

Figure 4. E-S histogram showing FRET, D-only and A-only populations. A 2-D ALEX histogram and marginal E and S histograms for a 40-bp dsDNA with D-A distance of 17 bases (Donor dye: ATTO550, Acceptor dye: ATTO647N). Bursts are selected with a size-threshold of 30 photons, including Aex photons. The plot is obtained with `alex_jointplot(ds)`. The 2D E-S distribution plot (join plot) is an histogram with hexagonal bins, which reduce the binning artifacts (compared to square bins) and naturally resembles a scatter-plot when the burst density is low. 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.

Another important parameter for defining the burst size is the γ-factor, i.e. the imbalance between the donor and the acceptor channel signals. 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 is significantly different from 1, neglecting its effect on burst size leads to over-representing (in terms of number of bursts) one FRET population versus the others.

When the γ factor is known, a more unbiased selection of different FRET populations can be achieved passing the argument `gamma` to the selection function:

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

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

For more details on burst size selection, see the `select_bursts.size` documentation (link).

Python details To compute γ-corrected burst sizes (with or without addition of `naa`) the method `Data.burst_sizes` (link) is used.

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, D-only and A-only bursts can be excluded by selecting bursts within 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.

An alternative approach consists in applying two selection filters sequentially. First, the A-only population is filtered out by applying a threshold on the number of photons during D excitation. Second, the D-only population is filtered out by applying a threshold on the number of photons during A excitation. The commands for these combined selections are:

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

Here, variable `ds2` contains the combined burst selection. Figure 5 shows the resulting pure FRET population obtained with the previous selection.

Figure 5. E-S histogram after filtering out D-only and A-only populations. 2-D ALEX histogram after selection of FRET population using the composition of two burst selection filters: (1) selection of bursts with counts in Dex stream larger than 15; (2) selection of bursts with counts in AexAem stream larger than 15. Compare to figure 4 where all burst populations (FRET, D-only and A-only) are reported.

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* [50], the underlying library used by FRETBursts to perform the fits.

Models can be created from arbitrary functions. By default, FRETBursts allows using predefined models such as 1 to 3 Gaussian peaks or 2-Gaussian connected by a "bridge". Built-in models are created by 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).

As an example, we fit the E histogram of bursts in the `ds` variable with two Gaussian peaks with 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 `binwidth` argument specifies the histogram bin width and the `model` argument defines which model shall be used for fitting.

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

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

Finally, to plot the fitted model together with the FRET histogram, as shown in figure 6, we pass the parameter `show_model=True` to the `hist_fret` function as follows (see section 7.4 for an introduction to plotting in FRETBursts):

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

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

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

Python details Models returned by FRETBursts's factory functions (`mfit.factory_*`) are `lmfit.Model` objects (link). Custom models can be created by calling `lmfit.Model` directly. When an `lmfit.Model` is fitted, it returns a `ModelResults` object (link), which contains all 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`. For instance, to obtain the reduced χ^2 value of the E histogram fit in a single-spot measurement `d`, we use the following command:

```
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). AIC and BIC allow comparing different models and selecting the most appropriate for the data at hand.

Example of definition and modification of fit models are provided in the aforementioned µs-ALEXnotebook. Users can also refer to the comprehensive lmfit's documentation (link).

5 Implementing Burst Variance Analysis

In this section, we describe how to implement burst variance analysis (BVA) as described in [23]. FRETBursts 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 the presence of several peaks clearly indicates the existence of multiple subpopulations, a single peak cannot a priori be associated with a single population defined by a unique FRET efficiency without further analysis (such as, for instance, shot-noise analysis [17, 18]).

The FRET histogram of a single FRET population has a minimum width set by shot noise (i.e. the width is caused by the statistics of discrete photon-detection events). FRET distributions broader than the shot noise limit, can be ascribed to either 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.). When the single peak of a FRET distribution is wider than predicted from shot-noise, it is not possible to discriminate between the static and dynamic case without further analysis. The BVA method has been developed to address this issue, namely identifying the presence of dynamics in FRET distributions [23], and has been successfully applied to identify biomolecular processes with dynamics on the millisecond time-scale [23, 51].

The basic idea behind BVA is to subdivide bursts into contiguous burst chunks (sub-bursts) comprising a fixed number n of photons, and to compare the empirical variance of acceptor counts across all sub-bursts in a burst with the theoretical shot-noise-limited variance, as expected from a binomial distribution. An empirical variance of sub-bursts larger than the shot-noise limited value indicates the presence of dynamics. 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 originating from a single static FRET efficiency, the sub-bursts acceptor counts n_a can be modeled as a binomial-distributed random variable $N_a \sim \mathrm{B}(n, E_p)$, where n is the number of photons in each sub-burst and E_p is the estimated population proximity-ratio. Note that we can use the PR because, regardless of the molecular FRET efficiency, the detected counts are partitioned between donor and acceptor channels according to a binomial distribution with success probability equal to the PR. The only approximation done here is neglecting the presence of 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 [23] for further discussion.

If N_a follows a binomial distribution, the random variable $E_{\mathrm{sub}} = N_a/n$, has a standard deviation reported in eq. 3.

$$\mathrm{Std}(E_{\mathrm{sub}}) = \left(\frac{E_p\,(1 - E_p)}{n}\right)^{1/2} \tag{3}$$

BVA analysis consists of four steps: 1) dividing bursts into consecutive sub-bursts containing a constant number of consecutive photons n, 2) computing the PR of each sub-burst, 3) calculating the empirical standard deviation (s_E) of sub-bursts PR in each 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 sub-populations (of different non-interconverting molecules) characterized by distinct FRET efficiencies, 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 belonging to a single specie, which interconverts between different FRET sub-populations (over times comparable to the diffusion time), as in figure 8, s_E of each burst will be larger than the expected shot-noise-limited standard deviation, and will be located above the shot-noise standard deviation curve (right panel of figure 8).

Figure 7. BVA distribution for a static mixture sample. The left panel shows the E-S histogram for a mixture of single stranded DNA (20dT) and double stranded DNA (20dT-20dA) molecules in 200 mM $MgCl_2$. The right panel shows the corresponding BVA plot. Since both 20dT and 20dT-20dA are stable and have no dynamics, the BVA plots shows s_E peaks lying on the static standard deviation curve (*red curve*).

Figure 8. BVA distribution for a hairpin sample undergoing dynamics. The left panel shows the E-S histogram for a single stranded DNA sample (A_{31}-TA, see in [52]), designed to form a transient hairpin in 400mM NaCl. The right panel shows the corresponding BVA plot. Since the transition between hairpin and open structure causes a significant change in FRET efficiency, s_E lies largely above the static standard deviation curve (*red curve*).

5.2 BVA Implementation

The following paragraphs describe the low-level details involved in implementing the BVA using FRETBursts. The main goal is to illustrate a real-world example of accessing

and manipulating timestamps and burst data. For a ready-to-use BVA implementation users can refer to the corresponding notebook included with FRETBursts (link).

Python details For BVA implementation, two photon streams are needed: all-photons during donor excitation (Dex) and acceptor photons during donor excitation (DexAem). These photon stream selections are obtained by computing boolean masks as follows (see section 7.3):

```
Dex_mask = ds.get_ph_mask(ph_sel=Ph_sel(Dex='DAem'))
DexAem_mask = ds.get_ph_mask(ph_sel=Ph_sel(Dex='Aem'))
DexAem_mask_d = AemDex_mask[Dex_mask]
```

Here, the first two variables (`Dex_mask` and `DexAem_mask`) select photon from the all-photons timestamps array, while `DexAem_mask_d`, selects A-emitted photons from the array of photons emitted during D-excitation. As shown below, the latter is needed to count acceptor photons in burst chunks.

Next, we need to express bursts start-stop data as indexes of the D-excitation photon stream (by default burst start-stop indexes refer to all-photons timestamps array):

```
ph_d = ds_FRET.get_ph_times(ph_sel=Ph_sel(Dex='DAem'))
bursts = ds_FRET.mburst[0]
bursts_d = bursts.recompute_index_reduce(ph_d)
```

Here, `ph_d` contains the Dex timestamps, `bursts` the original burst data and `bursts_d` the burst data with start-stop indexes relative to `ph_d`.

Finally, with the previous variables at hand, the BVA algorithm can be easily implemented by computing the s_E quantity for each burst:

```
n = 7
E_sub_std = []
for burst in bursts_d:
    E_sub = []
    startlist = range(burst.istart, burst.istop + 2 - n, n)
    stoplist = [i + n for i in startlist]
    for start, stop in zip(startlist, stoplist):
        A_D = DexAem_mask_d[start:stop].sum()
        E = A_D / n
        E_sub.append(E)
    E_sub_std.append(np.std(E_sub))
```

Here, `n` is the BVA parameter defining the number of photons in each burst chunk. The outer loop iterates through bursts, while the inner loop iterates through sub-bursts. The variables `startlist` and `stoplist` are the list of start-stop indexes for all sub-bursts in current burst. In the inner loop, `A_D` and `E` contain the number of acceptor photons and FRET efficiency for the current sub-burst. Finally, for each burst, the standard deviation of `E` is appended to the list `E_sub_std`.

By plotting the 2D distribution of s_E (i.e. `E_sub_std`) versus the average (uncorrected) E we obtain the BVA plots of figure 7 and 8.

6 Conclusions

FRETBursts is an open source and openly developed (see SI 7.2) implementation of established smFRET burst analysis methods made available to the single-molecule community. It implements several novel concepts which improve the analysis results, such as time-dependent background estimation, background-dependent burst search threshold, burst weighting and γ-corrected burst size selection. More importantly, FRETBursts provides a library of thoroughly-tested functions for timestamps and burst manipulation, making it an ideal platform for developing and comparing new analytical techniques.

We envision FRETBursts both as a state-of-the-art burst analysis software as well as a platform for development and assessment of novel algorithms. To underpin this envisioned role, FRETBursts is developed following modern software engineering practices, such as DRY principle (link) to reduce duplication and KISS principle (link) to reduce over-engineering. Furthermore, to minimize the number software errors [36, 53], we employ defensive programming [39] which includes code readability, unit and regression testing and continuous integration [28]. Finally, being open source, any scientist can inspect the source code, fix errors, adapt it to her own needs.

We believe that, in the single-molecule community, standard open source software implementations, such as FRETBursts, can enhance reliability and reproducibility of analysis and promote a faster adoption of novel methods, while reducing the duplication of efforts among different groups.

Acknowledgments

We thank Dr. Eyal Nir and Dr. Toma Tomov for support in the implementation of the 2CDE method. This work was supported by National Institutes of Health (NIH) grant R01-GM95904 and R01-GM069709. Dr. Weiss discloses equity in Nesher Technologies and intellectual property used in the research reported here. The work at UCLA was conducted in Dr. Weiss's Laboratory.

7 Supporting Information

7.1 Notebook Workflow

FRETBursts has been developed with the goal of facilitating computational reproducibility of the performed data analysis [25]. For this reason, the preferential way of using FRETBursts is by executing one of the tutorials which are in the form of Jupyter notebooks [35]. 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" [35] 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. A list of all FRETBursts notebooks can be found in the `FRETBursts_notebooks` repository on GitHub (link). Finally, we provide a service to run FRETBursts online, without requiring any software installation (link).

7.2 Development and Contributions

Errors are an inevitable reality in any reasonably complex software [36, 53]. It is therefore critical to implement countermeasures to minimize the probability of introducing bugs and their potential impact [37, 39]. We strove to follow modern software development best-practices, which are summarized below.

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 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 such as GitHub (link) for hosting, issues tracking and code reviews, TravisCI (link) and AppVeyor (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 at all, for a proprietary software or platform [54].

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 (with comments representing over 35% of the 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 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 continuous integration services, execute the full test suite on each commit on multiple platforms, immediately reporting errors. As a rule, whenever a bug is discovered, the fix also includes a new test to ensure that the same bug does not 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. The authors encourage users to use GitHub issues for questions, discussions and bug reports, and to submit patches through GitHub pull requests. Contributors of any level of expertise are welcome in the projects and publicly 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 become available to the entire single-molecule community.

7.3 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 (start and stop times, indexes and derived quantities for each burst) are stored in `Bursts` objects (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 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 FRETBursts 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 call `Data.get_ph_times` (link). Photon streams are selected from the full (all-photons) timestamps array using boolean masks, which can be obtained calling `Data.get_ph_mask` (link). All burst data (e.g. start-stop times and indexes, burst duration, etc.) are stored in `Bursts` objects. For uniformity, the bursts start-stop indexes are always referring to the all-photons timestamps array, regardless of the photon stream used for burst search. `Bursts` objects internally store only start and stop times and indexes. The other `Bursts` attributes (duration, photon counts, etc.) are computed on-the-fly when requested (using class properties), thus minimizing the object state. `Bursts` support iteration with performances similar to iterating through rows of 2D row-major numpy arrays.

7.4 Plotting `Data`

FRETBursts uses matplotlib [55] and seaborn [56] 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 multispot 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 which 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).

7.5 Background Estimation With Optimal Threshold

The functions used to fit the background (i.e. `bg.exp_fit` and other functions in `bg` module) provide also a goodness-of-fit estimator computed from the empirical distribution function (EDF) [57, 58]. 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 optimization is not necessary under typical experimental conditions, because the estimated rates normally change only a by a few per-cent compared to the heuristic threshold selection used by default.

7.6 Burst Weights

7.6.1 Theory

Freely-diffusing molecules across a Gaussian excitation volume give rise to a burst size distribution that is exponentially distributed. In a static FRET population, burst counts in the acceptor channel can be modeled as a binomial random variable (RV) with success probability equal to the population PR and number of trials equal to the burst size $n_d + n_a$. Similarly, the PR of each burst E_i (i being the burst index) is simply a binomial divided by the number of trials, with variance reported in eq. 4.

$$\mathrm{Var}(E_i) = \frac{E_p\,(1 - E_p)}{n_{ti}} \tag{4}$$

Bursts with higher counts, provide more accurate estimations of the population PR, since their PR variance is smaller (eq. 4). Therefore, in estimating the population PR we need to "focus" on bigger bursts. Traditionally, this is accomplished by merely discarding bursts below a size-threshold. In the following paragraphs we demonstrate how, by proper weighting bursts, is possible to obtains optimal estimates of PR which takes into account the information of the entire burst population.

According to the Cramer-Rao lower bound (eq. 5), the Fisher information $\mathcal{I}(\theta)$ sets a lower bound on the variance of any statistics $\hat{p}$ of a RV θ.

$$\mathrm{Var}\,(\hat{p}) \geq \frac{1}{\mathcal{I}(\theta)} \tag{5}$$

When the statistics $\hat{p}$ is an unbiased estimator of a distribution parameter and the equality holds in eq. 5, the estimator is a minimum-variance unbiased (MVUB) estimator and it is said to be efficient (meaning that it does an optimal use the information contained in the sample to estimate the parameter).

A population of N bursts can be modeled by a set of N binomial variables with same success probability E_p and varying number of successes equal to the burst size. An estimator for E_p can be constructed noticing that the sum of binomial RV with same success probability is still a binomial (with number of trials equal to the sum of the

number of trials). Taking the sum of acceptor counts over all bursts divided by the total number of photons as in eq. 6, we obtain an estimator $\hat{E}$ of the proportion of successes.

$$\hat{E} = \frac{\sum_i n_{ai}}{\sum_i n_{ti}} \tag{6}$$

The variance of $\hat{E}$ (eq. 7) is equal to the inverse of the Fisher information $\mathcal{I}(\hat{E})$ and therefore $\hat{E}$ is a MVUB estimator for E_p.

$$\mathrm{Var}(\hat{E}) = \frac{E_p(1-E_p)}{\sum_i n_{ti}} = \frac{1}{\mathcal{I}(\hat{E})} \tag{7}$$

We can finally verify that $\hat{E}$ is equal to the weighted average of the bursts PR E_i (eq. 9), with weights proportional to the burst size (eq. 8).

$$w_i = \frac{n_{ti}}{\sum_i n_{ti}} \tag{8}$$

$$\hat{E}_w = \frac{1}{N}\sum_i w_i E_i = \frac{1}{N}\frac{\sum_i n_{ti}\frac{n_{ai}}{n_{ti}}}{\sum_i n_{ti}} = \hat{E} \tag{9}$$

Since $\hat{E}$ is the MVUB estimator, any other estimator of E_p (in particular the unweighted mean of E_i) will have a larger variance.

We can extend these consideration of optimal weights for the PR estimator to the FRET distribution plot (histograms or KDEs). Building an unweighted histogram (and fitting the peak) is analogous to estimating the E_p with an unweighted average. Conversely, building the FRET histogram using the burst size as weights is equivalent to using the MVUB estimator for E_p.

7.6.2 Weighted FRET estimator

Here we report a simple verification of the results of previous section, namely that a weighted mean of E_i is the estimator with minimal variance of E_p. For this purpose, we generated a static FRET population of 100 bursts by simply extracting burst-sizes from an exponential distribution ($\lambda = 10$) and acceptor counts from a binomial distribution ($E_p = 0.2$). By repeatedly fitting the population parameter E_p using a size-weighted and unweighted average, we verified that the former has systematically lower variance of the latter as predicted by the theory (in the current example the unweighted estimator has 28.6 % higher variance). Note that this result holds for any arbitrary distribution of burst sizes. The full simulation including exponential and gamma-distributed burst sizes is reported in the accompanying Jupyter notebook (link).

7.6.3 Weighted FRET histogram

The effect of weighting the FRET histogram is here illustrated with a simulation of a mixture of two static FRET populations and then with experimental data.

We performed a realistic simulation of a static mixture of two FRET populations starting from 3-D Brownian motion diffusion of N particles excited by a numerically computed (non-Gaussian) PSF. Input parameters of the simulation include diffusion coefficient, particle brightness, the two FRET efficiencies, as well as detectors DCR. The simulation is performed with the open source software PyBroMo [41] which creates smFRET data files (i.e. timestamps and detectors arrays) in Photon-HDF5 format [40]. The simulated data file is processed with FRETBursts performing burst search, and only a minimal burst size selection of with threshold of 10 photons. The resulting

weighted and unweighted FRET histograms are reported in figure 9. We notice that the use of the weights results in better definition of FRET peaks.

As a final comparison, we report the weighted and unweighted FRET histogram of an experimental FRET population from measurement of a di-labeled dsDNA sample. Figure 10 show a comparison of a FRET histogram obtained from the same burst with and without weights. The burst selection is obtained applying a burst size threshold of 10 counts (after background correction), in order to filter the extreme low-end of the burst size distribution.

The use of size-weighted FRET histograms is a simple way to obtain a representation of FRET distribution that maintains high power of resolving FRET peaks while including the full burst population and thus reducing statistical noise.

As a final remark, note that when increasing the size-threshold for burst selection the difference between weighted and unweighted FRET histograms tends to zero because the relative difference in burst weights in the selected burst becomes smaller (i.e. tends to 1, meaning equal weights).

Figure 9. Comparison of unweighted and size-weighted FRET histograms for a simulated mixtures of static FRET populations. In both cases bursts are selected with a size threshold of 10 photons (after background correction).

Figure 10. Comparison of unweighted and size-weighted FRET histograms for a smFRET measurement of a static FRET sample (di-labeled dsDNA). In both cases bursts are selected with a size threshold of 10 photons (after background correction).

References

1. Weiss S. Fluorescence Spectroscopy of Single Biomolecules. Science. 1999;283(5408):1676–1683. doi:10.1126/science.283.5408.1676.
2. Hohlbein J, Craggs TD, Cordes T. Alternating-laser excitation: single-molecule FRET and beyond. Chem Soc Rev. 2014;43(4):1156–1171. doi:10.1039/c3cs60233h.
3. Lerner E, Orevi T, Ben Ishay E, Amir D, Haas E. Kinetics of fast changing intramolecular distance distributions obtained by combined analysis of FRET efficiency kinetics and time-resolved FRET equilibrium measurements. Biophysical journal. 2014;106(3):667–76. doi:10.1016/j.bpj.2013.11.4500.
4. Rahamim G, Chemerovski-Glikman M, Rahimipour S, Amir D, Haas E. Resolution of Two Sub-Populations of Conformers and Their Individual Dynamics by Time Resolved Ensemble Level FRET Measurements. PloS one. 2015;10(12):e0143732. doi:10.1371/journal.pone.0143732.
5. Selvin PR. The renaissance of fluorescence resonance energy transfer. Nat Struct Biol. 2000;7(9):730–734. doi:10.1038/78948.
6. Roy R, Hohng S, Ha T. A practical guide to single-molecule FRET. Nature Methods. 2008;5(6):507–516. doi:10.1038/nmeth.1208.
7. Schuler B, Eaton WA. Protein folding studied by single-molecule FRET. Current Opinion in Structural Biology. 2008;18(1):16–26. doi:10.1016/j.sbi.2007.12.003.

8. Sisamakis E, Valeri A, Kalinin S, Rothwell PJ, Seidel CAM. Accurate Single-Molecule FRET Studies Using Multiparameter Fluorescence Detection. In: Methods in Enzymology. Elsevier BV; 2010. p. 455–514. Available from: http://dx.doi.org/10.1016/S0076-6879(10)75018-7.

9. Haran G. How when and why proteins collapse: the relation to folding. Current Opinion in Structural Biology. 2012;22(1):14–20. doi:10.1016/j.sbi.2011.10.005.

10. Eggeling C, Fries JR, Brand L, Gunther R, Seidel CAM. Monitoring conformational dynamics of a single molecule by selective fluorescence spectroscopy. Proceedings of the National Academy of Sciences. 1998;95(4):1556–1561. doi:10.1073/pnas.95.4.1556.

11. Dahan M, Deniz AA, Ha T, Chemla DS, Schultz PG, Weiss S. Ratiometric measurement and identification of single diffusing molecules. Chemical Physics. 1999;247(1):85–106. doi:10.1016/s0301-0104(99)00132-9.

12. Fries JR, Brand L, Eggeling C, Köllner M, Seidel CAM. Quantitative Identification of Different Single Molecules by Selective Time-Resolved Confocal Fluorescence Spectroscopy. J Phys Chem A. 1998;102(33):6601–6613. doi:10.1021/jp980965t.

13. Eggeling C, Berger S, Brand L, Fries JR, Schaffer J, Volkmer A, et al. Data registration and selective single-molecule analysis using multi-parameter fluorescence detection. Journal of Biotechnology. 2001;86(3):163–180. doi:10.1016/s0168-1656(00)00412-0.

14. Zhang K, Yang H. Photon-by-Photon Determination of Emission Bursts from Diffusing Single Chromophores. J Phys Chem B. 2005;109(46):21930–21937. doi:10.1021/jp0546047.

15. Gopich I, Szabo A. Theory of photon statistics in single-molecule Förster resonance energy transfer. J Chem Phys. 2005;122(1):014707. doi:10.1063/1.1812746.

16. Lee NK, Kapanidis AN, Wang Y, Michalet X, Mukhopadhyay J, Ebright RH, et al. Accurate FRET Measurements within Single Diffusing Biomolecules Using Alternating-Laser Excitation. Biophysical Journal. 2005;88(4):2939–2953. doi:10.1529/biophysj.104.054114.

17. Nir E, Michalet X, Hamadani KM, Laurence TA, Neuhauser D, Kovchegov Y, et al. Shot-Noise Limited Single-Molecule FRET Histograms: Comparison between Theory and Experiments †. J Phys Chem B. 2006;110(44):22103–22124. doi:10.1021/jp063483n.

18. Antonik M, Felekyan S, Gaiduk A, Seidel CAM. Separating structural heterogeneities from stochastic variations in fluorescence resonance energy transfer distributions via photon distribution analysis. Journal of Physical Chemistry B. 2006;110(13):6970–6978. doi:10.1021/jp057257+.

19. Gopich IV, Szabo A. Single-Molecule FRET with Diffusion and Conformational Dynamics. J Phys Chem B. 2007;111(44):12925–12932. doi:10.1021/jp075255e.

20. Gopich IV. Concentration Effects in "Single-Molecule" Spectroscopy †. J Phys Chem B. 2008;112(19):6214–6220. doi:10.1021/jp0764182.

21. Camley BA, Brown FLH, Lipman EA. Förster transfer outside the weak-excitation limit. J Chem Phys. 2009;131(10):104509. doi:10.1063/1.3230974.

22. Santoso Y, Torella JP, Kapanidis AN. Characterizing Single-Molecule FRET Dynamics with Probability Distribution Analysis. ChemPhysChem. 2010;11(10):2209–2219. doi:10.1002/cphc.201000129.

23. Torella JP, Holden SJ, Santoso Y, Hohlbein J, Kapanidis AN. Identifying Molecular Dynamics in Single-Molecule FRET Experiments with Burst Variance Analysis. Biophysical Journal. 2011;100(6):1568–1577. doi:10.1016/j.bpj.2011.01.066.

24. Tomov TE, Tsukanov R, Masoud R, Liber M, Plavner N, Nir E. Disentangling Subpopulations in Single-Molecule FRET and ALEX Experiments with Photon Distribution Analysis. Biophysical Journal. 2012;102(5):1163–1173. doi:10.1016/j.bpj.2011.11.4025.

25. Buckheit JB, Donoho DL. WaveLab and Reproducible Research. In: Wavelets and Statistics. Springer Science + Business Media; 1995. p. 55–81. Available from: `http://dx.doi.org/10.1007/978-1-4612-2544-7_5`.

26. Ince DC, Hatton L, Graham-Cumming J. The case for open computer programs. Nature. 2012;482(7386):485–488. doi:10.1038/nature10836.

27. Vihinen M. No more hidden solutions in bioinformatics. Nature. 2015;521(7552):261–261. doi:10.1038/521261a.

28. Eglen S, Marwick B, Halchenko Y, Hanke M, Sufi S, Gleeson P, et al. Towards standard practices for sharing computer code and programs in neuroscience; 2016. Available from: `http://dx.doi.org/10.1101/045104`.

29. McKinney SA, Joo C, Ha T. Analysis of Single-Molecule FRET Trajectories Using Hidden Markov Modeling. Biophysical Journal. 2006;91(5):1941–1951. doi:10.1529/biophysj.106.082487.

30. Bronson JE, Fei J, Hofman JM, Gonzalez RL, Wiggins CH. Learning Rates and States from Biophysical Time Series: A Bayesian Approach to Model Selection and Single-Molecule FRET Data. Biophysical Journal. 2009;97(12):3196–3205. doi:10.1016/j.bpj.2009.09.031.

31. Greenfeld M, Pavlichin DS, Mabuchi H, Herschlag D. Single Molecule Analysis Research Tool (SMART): An Integrated Approach for Analyzing Single Molecule Data. PLoS ONE. 2012;7(2):e30024. doi:10.1371/journal.pone.0030024.

32. König SLB, Hadzic M, Fiorini E, Börner R, Kowerko D, Blanckenhorn WU, et al. BOBA FRET: Bootstrap-Based Analysis of Single-Molecule FRET Data. PLoS ONE. 2013;8(12):e84157. doi:10.1371/journal.pone.0084157.

33. van de Meent JW, Bronson JE, Wiggins CH, Gonzalez RL. Empirical Bayes Methods Enable Advanced Population-Level Analyses of Single-Molecule FRET Experiments. Biophysical Journal. 2014;106(6):1327–1337. doi:10.1016/j.bpj.2013.12.055.

34. Murphy RR, Jackson SE, Klenerman D. pyFRET: A Python Library for Single Molecule Fluorescence Data Analysis. ArXiv. 2014;.

35. Shen H. Interactive notebooks: Sharing the code. Nature. 2014;515(7525):151–152. doi:10.1038/515151a.

36. Soergel DAW. Rampant software errors may undermine scientific results. F1000Research. 2015;doi:10.12688/f1000research.5930.2.

37. Wilson G, Aruliah DA, Brown CT, Hong NPC, Davis M, Guy RT, et al. Best Practices for Scientific Computing. PLoS Biology. 2014;12(1):e1001745. doi:10.1371/journal.pbio.1001745.

38. Blischak JD, Davenport ER, Wilson G. A Quick Introduction to Version Control with Git and GitHub. PLOS Computational Biology. 2016;12(1):e1004668. doi:10.1371/journal.pcbi.1004668.

39. Prlić A, Procter JB. Ten Simple Rules for the Open Development of Scientific Software. PLoS Computational Biology. 2012;8(12):e1002802. doi:10.1371/journal.pcbi.1002802.

40. Ingargiola A, Laurence T, Boutelle R, Weiss S, Michalet X. Photon-HDF5: An Open File Format for Timestamp-Based Single-Molecule Fluorescence Experiments. Biophysical Journal. 2016;110(1):26–33. doi:10.1016/j.bpj.2015.11.013.

41. Ingargiola A, Laurence T, Boutelle R, Weiss S, Michalet X. Photon-HDF5: open data format and computational tools for timestamp-based single-molecule experiments. In: Enderlein J, Gregor I, Gryczynski ZK, Erdmann R, Koberling F, editors. Single Molecule Spectroscopy and Superresolution Imaging IX. SPIE-Intl Soc Optical Eng; 2016.Available from: http://dx.doi.org/10.1117/12.2212085.

42. Ingargiola A, Panzeri F, Sarkhosh N, Gulinatti A, Rech I, Ghioni M, et al. 8-spot smFRET analysis using two 8-pixel SPAD arrays. In: Enderlein J, Gregor I, Gryczynski ZK, Erdmann R, Koberling F, editors. Single Molecule Spectroscopy and Superresolution Imaging VI. SPIE; 2013.Available from: http://dx.doi.org/10.1117/12.2003704.

43. Kapanidis AN, Laurence TA, Lee NK, Margeat E, Kong X, Weiss S. Alternating-Laser Excitation of Single Molecules. Acc Chem Res. 2005;38(7):523–533. doi:10.1021/ar0401348.

44. Laurence TA, Kong X, Jäger M, Weiss S. Probing structural heterogeneities and fluctuations of nucleic acids and denatured proteins. PNAS. 2005;102(48):17348–17353. doi:10.1073/pnas.0508584102.

45. Müller BK, Zaychikov E, Bräuchle C, Lamb DC. Pulsed Interleaved Excitation. Biophysical Journal. 2005;89(5):3508–3522. doi:10.1529/biophysj.105.064766.

46. Michalet X, Colyer RA, Scalia G, Ingargiola A, Lin R, Millaud JE, et al. Development of new photon-counting detectors for single-molecule fluorescence microscopy. Philosophical Transactions of the Royal Society B: Biological Sciences. 2012;368(1611):20120035–20120035. doi:10.1098/rstb.2012.0035.

47. Edman L, Mets U, Rigler R. Conformational transitions monitored for single molecules in solution. Proceedings of the National Academy of Sciences. 1996;93(13):6710–6715. doi:10.1073/pnas.93.13.6710.

48. Deniz AA, Dahan M, Grunwell JR, Ha T, Faulhaber AE, Chemla DS, et al. Single-pair fluorescence resonance energy transfer on freely diffusing molecules: Observation of Forster distance dependence and subpopulations. Proceedings of the National Academy of Sciences. 1999;96(7):3670–3675. doi:10.1073/pnas.96.7.3670.

49. Kapanidis AN, Lee NK, Laurence TA, Doose S, Margeat E, Weiss S. Fluorescence-aided molecule sorting: Analysis of structure and interactions by alternating-laser excitation of single molecules. Proceedings of the National Academy of Sciences. 2004;101(24):8936–8941. doi:10.1073/pnas.0401690101.

50. Newville M, Stensitzki T, Allen DB, Ingargiola A. LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python¶; 2014. Available from: http://dx.doi.org/10.5281/zenodo.11813.

51. Robb NC, Cordes T, Hwang LC, Gryte K, Duchi D, Craggs TD, et al. 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. 2013;425(5):875–885. doi:10.1016/j.jmb.2012.12.015.

52. Tsukanov R, Tomov TE, Masoud R, Drory H, Plavner N, Liber M, et al. Detailed Study of DNA Hairpin Dynamics Using Single-Molecule Fluorescence Assisted by DNA Origami. The Journal of Physical Chemistry B. 2013;117(40):11932–11942. doi:10.1021/jp4059214.

53. Merali Z. Computational science: ...Error. Nature. 2010;467(7317):775–777. doi:10.1038/467775a.

54. Freeman J. Open source tools for large-scale neuroscience. Current Opinion in Neurobiology. 2015;32:156–163. doi:10.1016/j.conb.2015.04.002.

55. Droettboom M, Hunter J, Caswell TA, Firing E, Nielsen JH, Elson P, et al.. matplotlib: matplotlib v1.5.1; 2016. Available from: http://dx.doi.org/10.5281/zenodo.44579.

56. Waskom M, Botvinnik O, Hobson P, Warmenhoven J, Cole JB, Halchenko Y, et al.. seaborn: v0.6.0 (June 2015); 2015. Available from: http://dx.doi.org/10.5281/zenodo.19108.

57. Stephens MA. EDF Statistics for Goodness of Fit and Some Comparisons. Journal of the American Statistical Association. 1974;69(347):730. doi:10.2307/2286009.

58. Parr WC, Schucany WR. Minimum Distance and Robust Estimation. Journal of the American Statistical Association. 1980;75(371):616. doi:10.2307/2287658.