

Burst Variance Analysis: Help & Documentation

Kristofer Gryte¹

¹Department of Condensed Matter Physics, Clarendon
Laboratory, University of Oxford, Oxford, United Kingdom

October 21, 2011

Contents

Overview	1
Burst Variance Analysis	2
Installation	2
Input Arguments	3
Algorithm	4
Visualization	6
BVA Contour Plot	6
Efficiency Histogram	10
Standard Deviation Histogram	11
Concluding Remarks	12
Acknowledgments	13
Code	16
BVA Script	16
Initialize Axes	23
BVA Analysis	26
Data Filtering	45

CONTENTS

CONTENTS

Monte Carlo Simulation	54
Visualization	59

Overview

Burst variance analysis (BVA) attempts to detect dynamic fluctuations in Förster Resonance Energy Transfer (FRET) as individual molecules emitting donor and acceptor fluorescence diffuse through a confocal volume. Motivations for this method reside in the attempt to distinguish between static and dynamic heterogeneity and distribution broadening in confocal histograms. Existing methods proved unable to adequately address the static-dynamic distinction, thus necessitating a burst-by-burst analysis [1].

As detailed in Torella *et al*, BVA employs a statistical criterion to compare observed experimental standard deviations to simulated standard deviations generated according to theoretical prediction. Observed systems exceeding the strict confidence bounds are considered 'dynamic', where dynamics are assumed to arise due to state interconversion for a sample species. We note that BVA is not conclusive, however, in ascertaining whether observed dynamics arise due to conformational fluctuations, alone. Depending on experimental conditions and setup, artifacts originating from fluorescent dye quenching, dye isomerization, or optical misalignment can obfuscate underlying conformational dynamics, the presence or lack thereof.* On the other hand, what BVA can validate is distribution broadening due to species dynamics, whatever those 'dynamics' might be, as opposed to static heterogeneity, where sample species are in accordance with theoretical prediction. Hence, we urge that BVA be used as one of a suite of tools and experimental controls designed to effectively eliminate artifacts and determine system heterogeneity.

What follows is an overview of the BVA software package and its implementation. The creator of this software has endeavored to the best of his ability to ensure that the functions are well-documented and error-free. Should the user happen to be confused or receive any error messages, s/he is asked to consult the HELP documentation provided as well as the individual HELP sections of the functions provided herein. And should any programmatic mistakes be found, the user is asked to make a detailed note of the problem and report this to the author. Any feedback (comments, criticisms, or questions) is certainly welcome. The author hopes that this package accomplishes its aims and wishes the user the best in unraveling the mystery of single-molecule behavior. Enjoy!

*NOTE: efforts are currently underway within the Kapanidis laboratory to better address this issue and more clearly delineate conformational contributions.

Burst Variance Analysis

Installation

MATLAB® is the programming language for the BVA package. To install BVA,

- download the package either from the MATLAB® Central File Exchange or the laboratory webpage;
- save and unzip the *.zip file to a local directory;
- either add the path to the BVA package directory to `startup.m` (e.g., `addpath(C:\...\BVA)`), set the path from the MATLAB® command window (File → Set Path... → Add Folder), or navigate to the BVA folder within MATLAB® to make this the current directory;*
- once the top-level folder for BVA is accessible, the following functions may be run from the command-line:
 - `InitializeAxes.m`: configure figure axes.
 - `Confocal2ColorALExMultDataBVA_Analysis.m`: BVA analysis.
 - `PlotResults.m`: plot BVA results.
- to run the demo script `BurstVarianceAnalysis.m`, navigate to the BVA folder within MATLAB® to make this the current directory (sample data is contained in the file `MyArrivalMatrix.mat`) and type `BurstVarianceAnalysis` at the command-line;
- edit the demo script accordingly to analyze user data.

The BVA package is OS independent. At the present time, users need a STATS toolbox license to generate random numbers drawn from a binomial distribution. The author apologizes for any inconvenience this may cause.

*NOTE: only the top-level folder of the BVA package needs to be included in the path, as sub-folder access is controlled internally.

Input Arguments

BVA attempts to answer the question as to whether behavior consistent with ‘dynamic’ fluctuations is present for a sample species. Analysis is strictly dependent on two inputs: (i) a ‘colored’ photon *arrival matrix* and (ii) the specification of a *window* parameter. Additional input may be provided, such as filtering parameters (e.g, photon thresholds) and confidence interval adjustment (to either strengthen or weaken the barrier for fluctuations to be classified as dynamic). Default parameter values are built-in for all inputs except the data to be analyzed as well as filters/thresholds.* A full list of possible inputs is provided below:

- **ARRIVALMATRIX:** input data on which to perform BVA. The input matrix should be $N \times 3$ or $3 \times N$, where N equals the number of photons and with the following as vector inputs:
 1. Photon Arrival Times;
 2. Detection Channel (DexDem = 0; DexAem = 2; AexAem = 1; AexDem = 3);^{†‡}
 3. Burst Classification (those photons belonging to a burst should have an index corresponding to a unique burst ID; background photons should be indexed as NaN)[§]

The composition of the data matrix should be as follows:

$$\text{ARRIVALMATRIX} = \left[\text{ArrivalTimes} : \text{Color ID} : \text{BurstNumber} \right]$$

- **WINDOW:** size of the sliding window (number of photons) over which FRET will be calculated. Default: 5.
- **CLUSTERBINS:** number of bins over which to cluster bursts characterized by their mean FRET value. Mean standard deviations are calculated across bursts belonging to a particular bin and compared

*See Torella *et al* for an analysis on the effect of parameter values on dynamics detection. The default values are the same as those described elsewhere [1].

[†]Key: *D* = donor; *A* = acceptor; *ex* = excitation; *em* = emission

[‡]NOTE: we assume Alternating Laser Excitation (ALEX) spectroscopy [2], although single laser excitation of the donor is compatible.

[§]NOTE: burst classification presumes a burst search has already been performed. The included software package does not include burst search functionality. Needed information includes the START and END time for each burst. For information regarding the burst search method, consult [4].

against a test statistic to determine significance (and here, the presence of dynamics). Default: 20.

- **CLUSTERMINWINDOWS**: threshold for the minimum number of total windows within a cluster from all bursts of the cluster analyzed (i.e., we need to ensure statistical significance). Default: 50.
- **CONFIDENCEINTERVAL**: $(1-\alpha)$ to determine statistical significance of cluster results. Default: 0.999*.
- **FILTERS**: input structure providing minimum and maximum thresholds for the following data streams[†]:
 - **DURATION**: the temporal length of a detected burst [seconds]
 - **LENGTH**: the number of photons within a detected burst [photons]
 - **DEXDEM**: the number of photons detected in the donor channel upon donor excitation.
 - **DEXAEM**: the number of photons detected in the acceptor channel upon donor excitation.
 - **AEXAEM**: the number of photons detected in the acceptor channel upon acceptor excitation.
 - **SUMPHOTONS**: the number of photons detected in the donor & acceptor channels upon donor excitation.
 - **TOTALPHOTONS**: the number of photons detected in the donor & acceptor channels upon donor and acceptor excitation. (This excludes photons detected in donor channel upon acceptor excitation: AexDem.)
 - **EFFICIENCY**: the FRET efficiency.
 - **STOICHIOMETRY**: the relative fluorophore brightness.

Algorithm

The implementation of BVA is relatively straightforward:

*NOTE: the strict α value stems from a Bonferroni correction for multiple-hypothesis testing, which is dependent, here, on CLUSTERBINS. See [3] for its justification.

[†]NOTE: once again, we assume ALE_x; for single-laser excitation, ALE_x applicable fields may remain empty.

1. for each burst, bin the photons by arrival time into non-overlapping bins (bin size is defined by WINDOW).
2. calculate the FRET (E_j^*) for each bin.
3. calculate the standard deviation of E^* across all bins.
4. for all bursts with similar mean FRET $\{\mu^{E^*} : a \leq \mu^{E^*} < b\}$, calculate the cluster standard deviation across all bins for all satisfying bursts.
5. perform a hypothesis test to determine if the observed standard deviation exceeds confidence bounds for static behavior, thus being indicative of dynamics.

Pseudo-code is provided below:

Input: *ARRIVALMATRIX* size: $N \times 3$; N = total photons
Input: *WINDOW*
Output: *DATA*

Var: $C = \text{CLUSTERBINS}$; $\alpha = CI$; $W = \text{WINDOW}$

Filter bursts;

Partition $E^* \rightarrow \mu_{\{C\}}^{E^*} : [0 : 1/C : 1]$
(C_i = cluster index);

for $k := 1$ to N

◦ Extract photons for burst k from *ARRIVALMATRIX*

(L_k = total photons);

◦ Divide photons by arrival time into M_k non-overlapping windows
($M_k = L_k/W$);

for $j := 1$ to M_k

◦ Calculate $E_j^* = F_j^A/L$;

$$F^A = \sum_{i=1}^W (w_i == 2)$$

end

◦ Calculate $\mu_k^{E^*}$;

◦ Calculate $\sigma_k^{E^*}$;

◦ Determine $\mu_k^{E^*} \in \forall C_i$;

end

for $i := 1$ to C

```

    ◦  $\forall$  bursts satisfying  $\mu_k^{E*} \in C_i$ , calculate  $\mu_{C_i}^{\sigma^{E*}}$ ;
    ◦ Calculate shot-noise prediction:  $P(\sigma_{SN}^{E*})$ 
       $f(p, W, T); p = \mu_{C_i}^{E*}; T = \Sigma (M \forall \mu_k^{E*} \in C_i)$ ;
    ◦ Conduct multiple-hypothesis test ( $\alpha$ ) on  $P(\sigma_{SN}^{E*})$  for upper bound:  $UB_i$ ;
    ◦ Compare  $\mu_{C_i}^{\sigma^{E*}}$  to  $UB_i$  for significance;
end

```

$DATA \leftarrow RESULTS$;

Visualize $DATA$;

Visualization

BVA visualization is shown in Figure 1. The chosen format for graphical representation is an exploded plot having three components: (i) smoothed 2D histogram contour plot, (ii) projection of abscissa data (clustered FRET) in 1D histogram format, and (iii) projection of ordinate data (StDev FRET) in 1D histogram format.

BVA Contour Plot

BVA data (burst FRET mean versus standard deviation) is binned into a 100x100 bin grid, where the 2D bin size is determined by the minimum and maximum values along the respective dimensions. Obviously, this

Figure 1 (*following page*): Burst Variance Analysis (BVA) visualization. By plotting mean FRET versus standard deviation (StDev), a smoothed 2D histogram gives rise to the contour plot displayed. Red contours indicate areas of highest data density, whereas white represents zero density. The black line is the shot-noise prediction based on WINDOW. The dotted yellow line corresponds to the Monte Carlo shot-noise simulation based on experimental statistics. The Monte Carlo simulation and shot-noise prediction should agree. The pink line is the upper confidence bound for a static species. Triangles represent the mean values of the bursts grouped by partitioning the mean FRET axis. Black colored triangles are values falling below the upper confidence bound; whereas red colored triangles are values exceeding the upper bound and are indicative of dynamics. The colorbar displays the relative weighting of the colors used in the contour plot. The histograms are 1D projections along their respective axes. Further discussion is provided in the main text.



relative bin sizing should be remembered, particularly in comparing different data sets. Because the bin sizes are not fixed between samples, one cannot compare data sets absolutely—meaning data set 1 has more events at (x_1, y_1) than data set 2 has at the same location. What can be inferred is relative weighting within the respective data sets—meaning the relative weighting within data set 1 is similar in shape/distribution/et cetera to that in data set 2. The relative bin size approach is used for two reasons: (a) to allow more direct visual comparison between the relative distributions among data sets and (b) to accommodate the fact that the data distribution is not necessarily known beforehand. Whereas some prefer to think in bin sizes, others prefer to consider bin numbers. Here, we take a bin numbers approach, although the user is welcome to fix bin locations and size internally using the `histcn.m` function called in `plotresults.m`.*

Once the BVA is binned, we smooth the histogram using a spline smoothing function [5]. We have found a smoothing parameter of 20 yielded best visual results, balancing the trade-off between histogram noise and losing distinct histogram characteristics. This choice of default is dependent on data amount and quality. For those uncomfortable with a heuristic parameter, consult the `smoothn.m` documentation, which permits automatic parameter determination via generalized cross-validation. We do note, however, that consistency in smoothing parameter choice is more important than rigorous statistical validation, for, as long as the same smoothing parameter (chosen within reason) is applied across all samples (no analysis bias), any qualitative conclusions will remain valid.

The smoothed data yields the contour plot shown in the figure, where red represents high data density and white zero density. The colorbar values should not be considered as indicating absolute numbers of events, but be viewed in terms of relative data density.

Overlaid atop the contour plot are five data results from BVA:

1. a black parabolic curve representing the shot-noise prediction for static species, which is given by:

$$\sigma^{E^*} = \sqrt{\frac{E^*(1 - E^*)}{n}} \quad (1)$$

*For source code, see Bruno Luong, “N-dimensional histogram”, <http://www.mathworks.com/matlabcentral/fileexchange/23897>.

where $n = \text{WINDOW}$. Hence, in considering FRET alone, the shot-noise prediction provides a basis for visualizing the expected relation between μ^{E^*} and σ^{E^*} .

2. a yellow curve showing the mean of the Monte Carlo simulated standard deviation distribution. This curve serves as a sanity check that the simulation produced reliable results in accordance with theory and should provide reassurance in the event that the upper confidence bound yields results which do not conform to the user's expectations.
3. a pink data series displaying the upper confidence bound generated from a multiple hypothesis test [3].
4. filled black triangles represent μ^{E^*} and σ^{E^*} for the group (cluster) of bursts residing between two partitions and which have values consistent with static species. The edges of the cluster bin, assuming $\text{CLUSTERBINS} = 20$, are located ± 0.025 of μ^{E^*} .
5. filled red triangles represent μ^{E^*} and σ^{E^*} for the group (cluster) of bursts residing between two partitions and which have values inconsistent with static species and are indicative of dynamics. The edges of the cluster bin are the same as for the black triangles.

A few notes regarding data characteristics:

- one feature of the contour plot is the increasing spread observed in 2D distribution as μ^{E^*} trends toward 0.5 (emphasis on breath of blue dispersion). While heterogeneity is expected among molecules, as well as increased standard deviation as we move toward mid-FRET values, we would not expect increased heterogeneity (standard deviation) in the distribution of standard deviations. We would expect the standard deviation of standard deviations to remain constant, thus leading to a more or less constant spread as we trend toward 0.5.

We address this on two fronts: (i) recall that each individual burst is plotted separately, and, for the data presented, we did not explicitly apply a filter or threshold on burst length or total photons. Accordingly, the blue area is a region of relatively sparse events, many of which are bursts of low photon numbers (i.e., few windows over which to compute standard deviation). We can thus explain the

larger size of the blue area trending toward lower standard deviation by noting that, depending on the timescale and characteristics of dynamics, we expect that two neighboring non-overlapping windows have a greater probability of sharing a similar μ^{E*} value than two distant windows within the same burst. Hence, for bursts having only enough photons for, say, two or three windows (defined by WINDOW), we expect these bursts to exhibit lower standard deviations than ‘normal’. In the limit of infinite windows per burst, the contour distribution would exhibit zero standard deviation, but be a trending line. In which case, for this kind of analysis, the distribution spread says something about the timescale of dynamics. The information contained in these bursts regarding interconversion rates provides insight similar to auto-correlation analysis. (ii) Secondly, if we remove the blue contour in its entirety, we notice that the width of the contour distribution remains relatively constant, which is in accordance with our expectation.

- another feature is that the triangles do not consistently follow the weight of the contour distribution. With reference to the above, we should not necessarily expect these values to align with the distribution density, but rather the mean of the distribution spread within the partition. Hence, if we consider the two partitions centered at $\mu^{E*} = \{0.65, 0.70\}$, the standard deviation values are the means of the distributions within those partitions. These means do not necessarily equal the median values. Consequently, in the absence of the contour plot and assuming the interconversion of multiple states, the shape of the data series represented by triangles tells the user something about the timescale of dynamics.

Efficiency Histogram

Above the 2D histogram contour plot is a projection of the x-axis (cluster) data binned as a 1D histogram. We chose to bin the cluster data as opposed to the individual burst data to provide visual representation of the number of bursts summarized by the triangles in Figure 1. This information allows the user to better infer the data input for Monte Carlo shot-noise prediction and the associated upper confidence bound. Intuitively, the greater the support for a given cluster the more confident a user may be in determining the presence or absence of dynamics.

Standard Deviation Histogram

To the right of the 2D histogram contour plot is a projection of the y-axis data binned as a 1D histogram. In contrast to the x-axis projection, we chose to bin the individual burst data as opposed to the cluster data to provide visual representation of the underlying data structure supporting the contour plot in Figure 1. The 1D histogram serves as a reminder of the 100x100 bin grid and of the relative data density gradient represented by the colorbar. The cross-sections (areas under the curve) represented by the individual histogram bars along with knowledge of the shape of the distribution displayed in the x-axis histogram allow the user to infer approximately how many events support a given region of the contour plot.

Concluding Remarks

BVA can be an effective method to detect dynamics in single-molecule data. Used in conjunction with other analytical methods, such as probability distribution analysis (PDA) and correlation analysis, BVA provides a model-free method to hypothesis test for dynamics, helping eliminate those models, such as multiple static species, which yield similar distributions to that observed but are incompatible with fluctuations identified within individual bursts. We do note, however, that 'dynamics' is a broad term, including not only conformational fluctuations, but also photophysical effects resulting from quantum yield changes, dye sticking/stacking (restricted orientation), and other events which obfuscate biological behavior. Hence, from BVA alone, one cannot confidently attribute fluctuations solely to conformational dynamics. Appropriate experimental controls and optimization are needed to mitigate artifacts and conclusively explain distribution broadening. As such, BVA builds upon previous work investigating shot-noise limited distributions and provides a quick test for static versus dynamic heterogeneity as the origin of excess distribution width.

We hope this method and associated software prove beneficial in aiding single-molecule research, and we welcome any feedback (comments, criticisms, and questions) and input user's may have. Thank you for choosing to use BVA, and we look forward to your success in probing single-molecule behavior. All the best...

Acknowledgments

The author acknowledges the contributions of the following individuals who assisted in developing this software package, either in providing initial code or insights at various stages of development: Joseph Torella (Harvard University, Boston, USA), Johannes Hohlbein (University of Oxford, Oxford, UK), Yusdi Santoso (Boston Consulting Group, Indonesia), and many others who have tested and re-tested the code, providing feedback along the way. The author was supported by the Clarendon Fund (University of Oxford, Oxford, UK). The published work was supported by a European Commission Seventh Framework Programme (FP7/20072013) grant (HEALTH-F4-2008-201418, entitled READNA) and a Biotechnology and Biological Sciences Research Council grant (BB/H01795X/1) to Achillefs Kapanidis (University of Oxford, Oxford, UK).

References

- [1] Torella JP, Holden SJ, Santoso Y, Hohlbein J, Kapanidis AN (2011), "Identifying molecular dynamics in single-molecule FRET experiments with burst variance analysis", *Biophys J* 100(6):1568-77.
- [2] Kapanidis AN, Lee NK, Laurence TA, Doose S, Margeat E, Weiss S (2004) "Flourescence-aided molecule sorting: analysis of structure and interactions by alternating-laser excitation of single molecules", *PNAS* 101(24):8936-8941.
- [3] Abidi, H (2007), "Bonferroni and Sidak corrections for multiple comparisons", *Encyclopedia of Measurement and Statistics* Sage, Thousand Oaks (CA) 103-107.
- [4] Nir E, Michalet X, Hamadani KM, Laurence TA, Neuhauser D, Kovchegov Y, Weiss S (2006), "Shot-noise limited single-molecule FRET histograms: comparison between theory and experiments", *J Phys Chem B* 110(44):22103-22124.
- [5] Garcia, D (2009), "Robust smoothing of gridded data in one and higher dimensions with missing values", *Comput Stat Data An* 54:1167-1178. MATLAB code available at <http://www.mathworks.com/matlabcentral/fileexchange/25634>

List of Figures

1	Figure: Burst Variance Analysis	6
---	---	---

Code

BVA Script

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %% Burst Variance Analysis:
3 %
4 % Script:
5 %   This MATLAB script allows the user to perform Burst Variance Analysis
6 %   (BVA) and output this information to the user workspace and a figure.
7 %   Details concerning the BVA method and its application may be found in
8 %   the following publication:
9 %
10 %       Torella, JP, Holden, SJ, Santoso, Y, Hohlbein, J, Kapanidis, AN
11 %       (2011). Identifying molecular dyanmics in single-molecule FRET
12 %       experiments with burst variance analysis. Biophys J.
13 %       100(6):1568-77.
14 %
15 %   To run the script, set the Current Directory to the folder containing
16 %   the BVA package and run the script BURSTVARIANCEANALYSIS.M either from
17 %   the editor or from the command-line using the following command:
18 %   'BurstVarianceAnalysis '.
19 %
20 %   The test data is real experimental data. The input data to be analyzed
21 %   must meet two requirements:
22 %
23 %       [1] Individual photons are sorted by their arrival times and are
24 %       sorted according to a ALEx (see Kapanidis) color identification
25 %       scheme: (i) upon donor excitation (Dex): donor emission (Dem) -->
26 %       0, acceptor emission (Aem) --> 2; (ii) upon acceptor excitation
27 %       (Aex): donor emission (Dem) --> 3, acceptor emission (Aem) --> 1.
28 %       Thus, [DexDem, DexAem, AexAem, AexDem] --> [0, 2, 1, 3].
29 %
30 %       [2] A BURST SEARCH must have already been performed. Consult early
31 %       work out of the Siedel laboratory for this methodology. The two
32 %       most important outputs of the burst search are the start and end
33 %       times of the bursts. Such information can then be used to label
34 %       each photon as either belonging to a burst or not, and, if
35 %       belonging, to which burst it belongs.
36 %
37 %   Accordingly, the input data should be of the following format:
38 %
39 %       DATA = [ Arrival Times    Photon ID (0,2,1,3)    Burst ID]
40 %
41 %   As an example:
42 %       DATA = [ 0.40, 0, NaN;...
43 %                0.51, 1, NaN;...
```

```

44 %           0.52, 1, 1;...
45 %           0.53, 0, 1;...
46 %           0.54, 2, 1;...
47 %           0.55, 0, 1;...
48 %           0.62, 3, NaN;...
49 %           . . .
50 %           . . .
51 %           . . .
52 %       51.70, 2, NaN;...
53 %       51.74, 0, 100;...
54 %       51.75, 2, 100;...
55 %       51.76, 2, 100;...
56 %       51.77, 1, 100;...
57 %       51.78, 1, 100;...
58 %       52.81, 2, NaN];
59 %
60 % In the 'Testing' section, an example demonstrates how an Arrival Matrix
61 % containing only arrival times and photon IDs can be transformed to
62 % accommodate burst search input, as well as highlights some
63 % considerations regarding the data (e.g., long bursts being split into
64 % two, resulting in two adjacent bursts sharing the same photon, and the
65 % need to increasing end times by a fractional amount to ensure
66 % appropriate binning [NOTE: this, of course, can be avoided using
67 % a binning method other than HISTC.M]).
68 %
69 % Following the 'Testing' section, find the 'Initial Parameters' section,
70 % which provides a starting point for input parameters. The BVA parameters
71 % are typical values, as detailed in Torella, et al (2011); whereas, the
72 % filter parameters will vary depending on the experimental setup and
73 % conditions. For the example data included, the filter values are
74 % found appropriate.
75 %
76 % And finally, in the 'BVA' section, the figure generation, analysis, and
77 % plotting is broken down into 4 steps (functions), with example usage
78 % included.
79 %
80 % For additional questions and concerns regarding the individual
81 % functions themselves, please consult each function's HELP. Importantly,
82 % more detail concerning the nature and structure of the input and output
83 % parameters is provided. Further information concerning BVA
84 % interpretation may be found in the documentation provided: HELP.pdf .
85 %
86 % Lastly, if any bugs or breaks are found in the code, please contact the
87 % author of this package detailing the problem and provide any additional
88 % evidence of the error/problem, which includes, but is not limited to,
89 % error messages, figures, improper output data structure, et cetera. The
90 % author's contact information may be found below.
91 %
92 % NOTES:

```

```
93 %      [1] Need STATS toolbox for generating random numbers drawn from a
94 %      binomial distribution.
95 %
96 %      -----
97 %      Publications:
98 %      [1] Torella, JP, Holden, SJ, Santoso, Y, Hohlbein, J, Kapanidis, AN
99 %      (2011). Identifying molecular dyanmics in single-molecule FRET
100 %      experiments with burst variance analysis. Biophys J. 100(6):1568-77.
101 %
102 %      NOTICE:
103 %      If this software is used in the analysis of data which is to be
104 %      presented in publication, please cite the above work and
105 %      acknowledge the authors of this software.
106 %
107 %      For purposes of redistribution, use, and/or promotion of products
108 %      derived, consult the included license agreement.
109 %
110 %      Additionally, further license agreements are contained in
111 %      specialized subfunctions included in the directory 'Other'. Use,
112 %      redistribution, and derivative works are subject to any additional
113 %      requirements imposed therein.
114 %
115 %      Copyright (c) 2011. Kristofer Gryte. University of Oxford.
116 %
117 %
118 %
119 %      Version: 1.0 (2011-08-04)
120 %
121 %      -----
122 %      Author Information:
123 %      Kristofer Gryte
124 %      Clarendon Laboratory
125 %      University of Oxford
126 %      Oxford, United Kingdom
127 %      e-mail: k.gryte1 (at) physics.ox.ac.uk
128 %
129 %      -----
130 %      TODO:
131 %
132 %
133 %
134 %
135 %
136 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
137 %% EOP
138
139
140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
141 %% Testing:
```

```

142
143 addpath(...
144     'Other/lightspeed',...
145     'Other/SplitVec');
146
147 clear ArrivalMatrix StartTimes EndTimes
148
149 load('MyArrivalMatrix.mat');
150
151 % Check!!!! Before binning photons, need to ensure that no StartTimes
152 % and EndTimes are equal (i.e., no burst ends when another begins), as
153 % this creates problems when creating the edges (eliminates a non-burst
154 % bin).
155 AdjacentBursts = intersect_sorted(...
156     StartTimes,...
157     EndTimes);
158
159 if isempty(AdjacentBursts) == false
160     % Adjacent bursts have been found!!!
161     fprintf('%d bursts were found to be immediately adjacent.\n',...
162         numel(AdjacentBursts));
163
164     % Cycle through each Adjacent Burst:
165     for ADJBURST = 1 : numel(AdjacentBursts)
166
167         % Find the burst:
168         AdjBurstIndex = find(...
169             EndTimes == AdjacentBursts(ADJBURST),...
170             1,...
171             'first');
172
173         % Increase the arrival time of any photons which
174         % arrived at the previous start time by a fractional amount:
175         ArrivalMatrix(ArrivalMatrix(:,1) == EndTimes(AdjBurstIndex, 1), 1) ...
176             =...
177             ArrivalMatrix(ArrivalMatrix(:,1) == EndTimes(AdjBurstIndex, ...
178                 1), 1)...
179             +...
180             1e-10; % should be less than NI card time resolution...
181
182         % Increase the start time of the next burst by a fractional amount:
183         EndTimes(AdjBurstIndex, 1) =...
184             EndTimes(AdjBurstIndex, 1)...
185             +...
186             1e-10;
187
188     end % end FOR

```

```

189 end % end IF
190
191
192 % Configure the ArrivalMatrix:
193 Edges = union_sorted_rows(...
194     StartTimes,...
195     EndTimes + 1e-10); % Bump the end times by a fractional amount so that ...
                        % last photon can be placed in appropriate burst bin.
196
197 % Histogram the arrival times:
198 [Counts, BinIndices] = histc(ArrivalMatrix(:,1), Edges);
199
200 % Determine which bins to throw away: (e.g., all those bins containing
201 % photons which do not belong to a burst)
202
203 % Discard the even BinIndices, as these correspond to bins defined by
204 % EdgeA = EndTime(1); EdgeB = StartTime(2). Hence, this bin (EdgeA,
205 % EdgeB) corresponds to photons which do not belong to a burst (i.e., photons
206 % occurring after the end of the previous burst and before the start of
207 % the next burst):
208 VALS1 = 0:2:max(BinIndices); % Note: we include 0 to account for photons ...
                        % not falling into any defined bin.
209 TF = ismember_sorted(BinIndices, VALS1);
210
211
212 % Remove all photons not belonging to a burst, thus creating a 'burst
213 % arrival matrix':
214 BinIndices(TF == true) = NaN; %[];
215
216 % Tack on the BinIndices (i.e., Burst Indices) to the Arrival Matrix:
217 ArrivalMatrix(:,3) = BinIndices;
218
219 clear BinIndices TF VALS1 Counts Edges
220
221
222 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
223 %% Initial Parameters:
224
225 WINDOW = 5;
226 CLUSTERBINS = 20;
227 CLUSTERMINWINDOWS = 50;
228 CONFIDENCEINTERVAL = 0.999;
229
230 % ----- %
231 % Filters:
232 % ----- %
233
234 % DexDem:
235 FILTERS.DexDem.Min = [];

```

```

236 FILTERS.DexDem.Max = [];
237
238 % DexAem:
239 FILTERS.DexAem.Min = [];
240 FILTERS.DexAem.Max = [];
241
242 % AexAem:
243 FILTERS.AexAem.Min = [30];
244 FILTERS.AexAem.Max = [];
245
246 % Sum Photons: DexDem + DexAem
247 FILTERS.SumPhotons.Min = [];
248 FILTERS.SumPhotons.Max = [];
249
250 % Total Photons: DexDem + DexAem + AexAem
251 FILTERS.TotalPhotons.Min = [];
252 FILTERS.TotalPhotons.Max = [];
253
254 % Efficiency:
255 FILTERS.Efficiency.Min = [];
256 FILTERS.Efficiency.Max = [];
257
258 % Stoichiometry:
259 FILTERS.Stoichiometry.Min = [0.45];
260 FILTERS.Stoichiometry.Max = [];
261
262 % Duration: [seconds]
263 FILTERS.Duration.Min = [];
264 FILTERS.Duration.Max = [];
265
266 % Length: [photons]
267 FILTERS.Length.Min = [];
268 FILTERS.Length.Max = [];
269
270
271
272 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
273 %% BVA:
274
275 % [1] Create a figure:
276 handles.figure = figure;
277
278 set(handles.figure,...
279     'Position', [520, 196, 900, 700]); % [x-pos, y-pos, width, height] % ...
    NOTE: the specified values are appropriate for the INITIALIZEAXES ...
    function to follow. Should the figure window be too large or ...
    small, subsequent re-adjustments in the INITIALIZEAXES function ...
    may be needed.
280

```

```
281 % [2] Initialize the figure and axes:
282 handles = InitializeAxes(handles);
283
284 % [3] Perform BVA:
285 [DATA] = Confocal2ColorALExSolnMultDataBVA_Analysis(...
286     ArrivalMatrix,...
287     'window', WINDOW,...
288     'clusterbins', CLUSTERBINS,...
289     'clusterminwindows', CLUSTERMINWINDOWS,...
290     'confidenceinterval', CONFIDENCEINTERVAL,...
291     'filters', FILTERS);
292
293 % [4] Plot the results:
294 handles = PlotResults(DATA, WINDOW, handles);
295
296
297 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
298 %% EOS
```


Initialize Axes

```

1 function handles = InitializeAxes(handles)
2 % INITIALIZEAXES
3 %   Initialize axes within a figure window according to defined
4 %   specifications. Three subplots are generated: [i] 2D histogram subplot
5 %   and [ii] 2 1D histogram subplots along the upper and right edges of the
6 %   2D histogram subplot. Orientations of the 1D histogram subplots are
7 %   such that they are the projections of the data along the abscissa and
8 %   ordinate axes, respectively.
9 %
10 % INPUTS:
11 %     HANDLES: structure with the following fields:
12 %         FIGURE: handle to the figure window. Suggested figure size:
13 %         [520, 196, 900, 700].
14 %
15 %
16 % OUTPUTS:
17 %     HANDLES: structure with the following fields:
18 %         SUBPLOT: nested structure with the following fields:
19 %             BVA: nested structure with the following fields:
20 %                 AXES: handles to the 2D Histogram axes (subplot).
21 %         SUBPLOT: nested structure with the following fields:
22 %             HISTOGRAM: nested structure with the following fields:
23 %                 MEAN: nested structure with the following fields:
24 %                     AXES: handle to the axes into which the histogram
25 %                         will be plotted.
26 %                 STDEV: nested structure with the following fields:
27 %                     AXES: handle to the axes into which the histogram
28 %                         will be plotted.
29 %
30 %
31 %
32 % DEPENDENCIES:
33 %     (none)
34 %
35 % -----
36 % EXAMPLE USAGE:
37 %     handles = InitializeAxes(handles);
38 %
39 % -----
40 % History:
41 %     [1] Kristofer Gryte. University of Oxford (2011).
42 %
43 % Copyright (c) 2011. Kristofer Gryte. University of Oxford.
44 %
45 %

```

```

46 %
47 %   Version: 1.0 (2011-08-02)
48 %
49 %   -----
50 %   Author Information:
51 %       Kristofer Gryte
52 %       Clarendon Laboratory
53 %       University of Oxford
54 %       Oxford, United Kingdom
55 %       e-mail: k.gryte1 (at) physics.ox.ac.uk
56 %
57 %   -----
58 %   TODO:
59 %
60 %
61
62 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
63 %% EOP
64
65
66 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
67 %% Initialization:
68
69 % Activate figure window:
70 figure(handles.figure);
71
72 % Configure the figure colormap:
73 load('BVAc colormap.mat', 'BVAc colormap');
74
75 set(handles.figure,...
76     'Colormap', BVAc colormap);
77
78
79 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
80 %% Initialize Subplots:
81
82 % Mean Histogram Subplot:
83 handles.Subplot.Histogram.Mean.Axes = subplot(3,1,1);
84
85 % Standard Deviation Histogram Subplot:
86 handles.Subplot.Histogram.StDev.Axes = subplot(3,1,2);
87
88 % BVA Subplot:
89 handles.Subplot.BVA.Axes = subplot(3,1,3);
90
91
92
93 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
94 %% Manipulate Subplots:

```

```

95
96 % Get the current units:
97 OldUnits = get(handles.Subplot.Histogram.Mean.Axes, 'Units');
98
99 % Set Units: % Note: Change to characters...as the number of pixels is
100 % relative to user monitor...thus, standardize...
101 set([handles.Subplot.Histogram.Mean.Axes,...
102     handles.Subplot.Histogram.StDev.Axes,...
103     handles.Subplot.BVA.Axes],...
104     'Units', 'pixels');
105
106 % Set Mean Histogram Position:
107 set(handles.Subplot.Histogram.Mean.Axes,...
108     'Position', [150 500 350 150]);
109
110 % Set StDev Histogram Position:
111 set(handles.Subplot.Histogram.StDev.Axes,...
112     'Position', [600 95 150 350]);
113
114 % Set BVA Subplot Position:
115 set(handles.Subplot.BVA.Axes,...
116     'Position', [150 95 350 350]);
117
118 % Return units to 'characters':
119 set([handles.Subplot.Histogram.Mean.Axes,...
120     handles.Subplot.Histogram.StDev.Axes,...
121     handles.Subplot.BVA.Axes],...
122     'Units', OldUnits); % Maybe Normalize?
123
124
125 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
126 %% EOF

```

BVA Analysis

```

1 function [DATA] = Confocal2ColorALEXSolnMultDataBVA_Analysis(...)
2     varargin)
3 % CONFOCAL2COLORALEXSOLNMULTDATABVA_ANALYSIS
4 % Burst Variance Analysis (BVA) of confocal (2 color) solution
5 % alternating laser excitation (ALEX) data.
6 %
7 % INPUTS:
8 %     ARRIVALMATRIX: input data on which to perform Burst Variance
9 %     Analysis. The input matrix should be Nx3 or 3xN, with the following
10 %    as vector inputs: (1) Photon Arrival Times; (2) Detection Channel
11 %    (DexDem = 0; DexAem = 2; AexAem = 1; AexDem = 3); (3) Burst
12 %    Classification (those photons belonging to a burst should have a
13 %    corresponding index to a unique burst ID; background photons should
14 %    be indexed as NaN): ARRIVALMATRIX = ArrivalTimes    ID    BurstNumber
15 %
16 %     WINDOW: size of the sliding window over which FRET will be
17 %    calculated. Default: 5.
18 %
19 %     CLUSTERBINS: number of bins over which to cluster bursts
20 %    characterized by their mean FRET value. Mean standard deviation
21 %    will be calculated across bursts belonging to a particular bin and
22 %    compared against a test statistic to determine significance (and
23 %    here, the presence of dynamics). Default: 20.
24 %
25 %     CLUSTERMINWINDOWS: threshold for the minimum number of total windows
26 %    within a cluster from all bursts of the cluster to be analyzed
27 %    (i.e., we need to make sure we have enough data to be worth our
28 %    while). Default: 50.
29 %
30 %     CONFIDENCEINTERVAL: (1-alpha) to determine statistical significance
31 %    of cluster results. Default: 0.999.
32 %
33 %     FILTERS: nested structure providing minimum and maximum thresholds
34 %    for the following data streams:
35 %         DURATION: the temporal length of a detected burst [seconds]
36 %         LENGTH: the number of photons within a detected burst [photons]
37 %         DEXDEM: the number of photons detected in the donor channel
38 %             upon donor excitation.
39 %         DEXAEM: the number of photons detected in the acceptor channel
40 %             upon donor excitation.
41 %         AEXAEM: the number of photons detected in the acceptor channel
42 %             upon acceptor excitation.
43 %         SUMPHOTONS: the number of photons detected in the donor &
44 %             acceptor channels upon donor excitation.
45 %         TOTALPHOTONS: the number of photons detected in the donor &

```

```

46 %         acceptor channels upon donor and acceptor excitation.
47 %         (excluding photons detected in donor channel upon acceptor
48 %         excitation: AexDem)
49 %     EFFICIENCY: the FRET efficiency.
50 %     STOICHIOMETRY: the relative fluorophore brightness.
51 %
52 % OUTPUTS:
53 %     DATA: structure with the following data streams for each detected
54 %     burst: (NOTE: data streams are filtered according to input FILTERS)
55 %         DURATION: the temporal length of a detected burst [seconds]
56 %         LENGTH: the number of photons within a detected burst [photons]
57 %         DEXDEM: the number of photons detected in the donor channel
58 %         upon donor excitation.
59 %         DEXAEM: the number of photons detected in the acceptor channel
60 %         upon donor excitation.
61 %         AEXAEM: the number of photons detected in the acceptor channel
62 %         upon acceptor excitation.
63 %         SUMPHOTONS: the number of photons detected in the donor &
64 %         acceptor channels upon donor excitation.
65 %         TOTALPHOTONS: the number of photons detected in the donor &
66 %         acceptor channels upon donor and acceptor excitation.
67 %         (excluding photons detected in donor channel upon acceptor
68 %         excitation: AexDem)
69 %         EFFICIENCY: the FRET efficiency.
70 %         STOICHIOMETRY: the relative fluorophore brightness.
71 %         RESULTS: an Nx2 array, where N is the number of bursts (after
72 %         filtering). The first column corresponds to the mean ...
Efficiency
73 %         value for a burst and the second to the burst's Efficiency
74 %         standard deviation. Array: E(efficiency)  sqrt(V(efficiency))
75 %     CLUSTERS: nested structure with the following fields:
76 %         CENTERS: vector containing the centers of the individual
77 %         clusters.
78 %         STDEVDISTRMEAN: the Monte Carlo simulated standard
79 %         deviation prediction.
80 %         STATISTICS: an Mx2 array, where M is the number of clusters
81 %         ('slices' along the efficiency axis, into which bursts are
82 %         grouped). The first column corresponds to the mean
83 %         Efficiency value for the cluster, and the second to the
84 %         standard deviation of the burst's within that cluster.
85 %         WINDOWMEANS: cellular array where each entry contains the
86 %         the Efficiency values for all sliding windows for bursts
87 %         belonging to a cluster.
88 %         DISTANCEFROMUPPERCI: a vector of the distance between the
89 %         mean standard deviations of each cluster from the upper
90 %         confidence bound. This expresses something about the ...
extent
91 %         of the 'dynamics'.
92 %     CONFIDENCEINTERVAL: nested structure with the following fields:

```

```

93 %           UPPER: the upper confidence bound generated from the Monte
94 %               Carlo simulation and multiple hypothesis test.
95 %           LOWER: the lower confidence bound generated from the Monte
96 %               Carlo simulation and the multiple hypothesis test.
97 %
98 % DEPENDENCIES:
99 %     SPLITVEC.M
100 %     INTERSECT_SORTED.M (Lightspeed)
101 %     UNION_SORTED_ROWS.M (Lightspeed)
102 %     ISMEMBER_SORTED.M (Lightspeed)
103 %     SETDIFF_SORTED.M (Lightspeed)
104 %     FILTERDATA.M (private)
105 %     MONTECARLOPREDICTION.M (private; requires STATS toolbox)
106 %
107 % -----
108 % EXAMPLE USAGE:
109 %     Confocal2ColorALExSolnMultDataBVA_Analysis(ArrivalMatrix);
110 %
111 %     Confocal2ColorALExSolnMultDataBVA_Analysis(ArrivalMatrix, 'window',
112 %         5);
113 %
114 %     Confocal2ColorALExSolnMultDataBVA_Analysis(ArrivalMatrix, 'window',
115 %         5, 'clusterbins', 20, 'clusterminwindows', 50);
116 %
117 % -----
118 % History:
119 %     [1] Joseph Torella, Yusdi Santoso. University of Oxford (2009).
120 %     [2] Johannes Hohlbein. University of Oxford (2010).
121 %     [3] Kristofer Gryte. University of Oxford (2011).
122 %
123 % Publications:
124 %     [1] Torella, JP, Holden, SJ, Santoso, Y, Hohlbein, J, Kapanidis, AN
125 %         (2011). Identifying molecular dyanmics in single-molecule FRET
126 %         experiments with burst variance analysis. Biophys J. 100(6):1568-77.
127 %
128 % NOTICE:
129 %     If this software is used in the analysis of data which is to be
130 %     presented in publication, please cite the above work and
131 %     acknowledge the authors of this software.
132 %
133 % Copyright (c) 2011. Kristofer Gryte. University of Oxford.
134 %
135 %
136 %
137 % Version: 1.0 (2011-07-29)
138 %
139 % -----
140 % Author Information:
141 %     Kristofer Gryte

```

```

142 %      Clarendon Laboratory
143 %      University of Oxford
144 %      Oxford, United Kingdom
145 %      e-mail: k.grytel (at) physics.ox.ac.uk
146 %
147 %      -----
148 %      TODO:
149 %          [1] Provide appropriate checks on filter values (e.g., ensure
150 %              numeric, positive, etc.)
151 %          [2] Make number of Monte Carlo simulations an input variable?
152 %
153 %      -----
154 %      See also
155 %
156 %
157
158 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
159 %% EOP
160
161
162
163 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
164 %% StartUp/CleanUp:
165
166 StartUp();
167
168 C = onCleanup(@() Cleanup());
169
170
171 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
172 %% Initialization:
173
174 % ---- %
175 % Establish default values:
176 % ---- %
177 WINDOW = 5; % size of the sliding window (non-overlapping) over which FRET ...
178             % will be calculated; hence, for every 5 photons, we calculate FRET
179 CLUSTERBINS = 20; % number of bins over which we will cluster bursts ...
180             % characterized by their mean FRET value
181 CLUSTERMINWINDOWS = 50; % threshold for the minimum number of total ...
182             % windows within a cluster over all bursts for the cluster to be ...
183             % analyzed (i.e., we need to make sure we have enough data to be worth ...
184             % our while)
185
186 CONFIDENCEINTERVAL = 0.999; % (1-alpha) CI level to determine if observed ...
187             % variance is statistically significant
188 NUMMONTECARLOSAMPLES = 500; % Number of Monte Carlo simulations to run for ...
189             % CI bound generation
190
191

```

```

184 FILTERS = struct(...
185     'DexDem',      struct('Min', [], 'Max', []),...
186     'DexAem',      struct('Min', [], 'Max', []),...
187     'AexAem',      struct('Min', [], 'Max', []),...
188     'AexDem',      struct('Min', [], 'Max', []),...
189     'Efficiency',  struct('Min', [], 'Max', []),...
190     'Stoichiometry', struct('Min', [], 'Max', []),...
191     'SumPhotons',  struct('Min', [], 'Max', []),...
192     'TotalPhotons', struct('Min', [], 'Max', []),...
193     'Duration',    struct('Min', [], 'Max', []),...
194     'Length',      struct('Min', [], 'Max', []));
195
196 DATA = [];
197
198 % ---- %
199 % Parse input arguments:
200 % ---- %
201
202 ArrivalMatrix = varargin{1};
203
204 % Check!!!
205 if ~isnumeric(ArrivalMatrix)
206     % error out!
207     errordlg(['ERROR:invalid input argument for BVA function. Please input ...
208             ',...
209             'a numeric array for the initial argument.'],...
210             'ERROR:invalid input argument');
211     return;
212 end % end IF
213
214 % Perform check for data input size and ensure dimensionality is correct
215 % (column matrix)
216 if size(ArrivalMatrix,1) < size(ArrivalMatrix, 2) % More columns than rows
217     % Take the tranpose:
218     ArrivalMatrix = ArrivalMatrix';
219 elseif size(ArrivalMatrix,2) ≠ 3 % too many or too few columns
220
221     % Error out!
222     errordlg(['ERROR:invalid input argument for BVA function. Data input ',...
223             'should be either (Nx3) or (3xN).'],...
224             'ERROR:invalid input argument');
225     return;
226
227 end % end IF/ELSEIF
228
229 % Check number of input arguments:
230 if (mod(nargin, 2) - 1) ≠ 0 % odd number of input arguments
231     % Throw a fit and error out!

```



```

232
233     errordlg(['ERROR:invalid number of input arguments for BVA function. ',...
234             'Please enter parameter-value pairs in addition to the data ...
                input.'],...
235             'ERROR:invalid number of input arguments');
236     return;
237
238 end % end IF
239
240 % Loop through and assign parameter-value pairs:
241 for NumArg = 2 : 2 : nargin
242
243     switch lower(char(varargin{NumArg}))
244
245         case {'win', 'window', 'winsize'}
246             % Update the WINDOW size:
247             temp = varargin{NumArg + 1};
248
249             % Check!
250             if ~isnumeric(temp)    temp ≤ 1
251                 % Error out:
252                 answer = questdlg(...
253                     ['NOTICE:An invalid value was provided for the WINDOW ...
                        paramter. ',...
254                     'Please supply an integer value greater than 1. ',...
255                     'Would you like to continue with a default WINDOW = 5, ...
                        ',...
256                     'or return to the command-line and supply an ...
                        appropriate input value?'],...
257                     'WARNING:invalid input value',...
258                     'Continue', 'Cancel',...
259                     'Cancel');
260
261                 % Parse the answer:
262                 switch lower(answer)
263                     case 'continue'
264                         % Move along with default WINDOW value
265                     case 'cancel'
266                         % Abort!
267                         return;
268                 end % end SWITCH
269
270             else
271                 % Assign over the input WINDOW size:
272                 WINDOW = temp;
273             end % end IF/ELSE
274
275         case {'clusterbins', 'bins', 'numbins', 'binnumber', 'binnum'}
276

```

```

277     % Update the number of cluster bins:
278     temp = varargin{NumArg + 1};
279
280     % Check!
281     if ~isnumeric(temp)    temp ≤ 1
282         % Error out:
283         answer = questdlg(...
284             ['An invalid value was provided for the CLUSTERBINS ...
285              paramter. ',...
286              'Please supply an integer value greater than 1. ',...
287              'Would you like to continue with a default CLUSTERBINS ...
288               = 20, ',...
289              'or return to the command-line and supply an ...
290               appropriate input value?'],...
291              'WARNING:invalid input value',...
292              'Continue', 'Cancel',...
293              'Cancel');
294
295         % Parse the answer:
296         switch lower(answer)
297             case 'continue'
298                 % Move along with default CLUSTERBINS value
299             case 'cancel'
300                 % Abort!
301                 return;
302         end % end SWITCH
303
304     else
305         % Assign over the input CLUSTERBINS value:
306         CLUSTERBINS = temp;
307     end % end IF/ELSE
308
309     case {'clusterminwindows', 'minwindows'}
310
311         % Update the minimum number of windows within a cluster:
312         temp = varargin{NumArg + 1};
313
314         % Check!
315         if ~isnumeric(temp)    temp ≤ 1
316             % Error out:
317             answer = questdlg(...
318                 ['An invalid value was provided for the ...
319                  CLUSTERMINWINDOWS paramter. ',...
320                  'Please supply an integer value greater than 1. ',...
321                  'Would you like to continue with a default ...
322                   CLUSTERMINWINDOWS = 50, ',...
323                  'or return to the command-line and supply an ...
324                   appropriate input value?'],...
325                  'WARNING:invalid input value',...

```

```

320         'Continue', 'Cancel', ...
321         'Cancel');
322
323     % Parse the answer:
324     switch lower(answer)
325     case 'continue'
326         % Move along with default CLUSTERMINWINDOWS value
327     case 'cancel'
328         % Abort!
329         return;
330     end % end SWITCH
331
332     else
333         % Assign over the input CLUSTERMINWINDOWS:
334         CLUSTERMINWINDOWS = temp;
335     end % end IF/ELSE
336
337     case {'ci', 'confidenceinterval', 'conf', 'confidence', ...
338          'confinterval'}
339
340         % Update the confidence interval:
341         temp = varargin{NumArg + 1};
342
343         % Check!
344         if ~isnumeric(temp)    (temp ≥ 1    temp ≤ 0)
345             % Error out:
346             answer = questdlg(...
347                 ['An invalid value was provided for the ...
348                  CONFIDENCEINTERVAL paramter. ', ...
349                  'Please supply a numeric value greater than 0 and less ...
350                  than 1. ', ...
351                  'Would you like to continue with a default ...
352                  CONFIDENCEINTERVAL = 0.999, ', ...
353                  'or return to the command-line and supply an ...
354                  appropriate input value?'], ...
355                  'WARNING:invalid input value', ...
356                  'Continue', 'Cancel', ...
357                  'Cancel');
358
359             % Parse the answer:
360             switch lower(answer)
361             case 'continue'
362                 % Move along with default CONFIDENCEINTERVAL value
363             case 'cancel'
364                 % Abort!
365                 return;
366             end % end SWITCH
367
368         else

```

```

364         % Assign over the input CONFIDENCEINTERVAL:
365         CONFIDENCEINTERVAL = temp;
366     end % end IF/ELSE
367
368     case {'filters', 'filter', 'thresh', 'thresholds', 'threshold', ...
369          'filt'}
370
371         % Update the FILTERS:
372         temp = varargin{NumArg + 1};
373
374         % Check!
375         if ~isstruct(temp)
376             % Error out:
377             answer = questdlg(...
378                 ['An invalid value was provided for the FILTERS ...
379                  paramter. ',...
380                  'Please supply an appropriate structure containing ...
381                  relevant fields. ',...
382                  'Would you like to continue with a default FILTERS ...
383                  (none), ',...
384                  'or return to the command-line and supply an ...
385                  appropriate input?'],...
386                  'WARNING:invalid input value',...
387                  'Continue', 'Cancel',...
388                  'Cancel');
389
390             % Parse the answer:
391             switch lower(answer)
392                 case 'continue'
393                     % Move along with default FILTERS value
394                 case 'cancel'
395                     % Abort!
396                     return;
397             end % end SWITCH
398
399         else
400             % Assign over the input FILTERS:
401             FILTERS = temp;
402         end % end IF/ELSE
403
404     otherwise
405         % Throw a notice that the input argument was not recognized:
406         msgbox(sprintf(...
407             ['NOTICE:Unrecognized input argument: %s. \n',...
408              'Parameter-value pair ignored']),...
409             char(varargin{NumArg}));
410     end % end SWITCH

```

```

408 end % end FOR
409
410
411 % ---- %
412 % Calculate remaining parameters:
413 PHOTONCUTOFF = floor(WINDOW/2); % Remove the first few photons of a burst ...
    (allows analyzed data to be centered within burst)
414
415 CLUSTERWIDTH = (1-0)/CLUSTERBINS; % How wide are our bins in which we are ...
    clustering bursts (UNITS: none --> FRET efficiency); (0,1) defines the ...
    range of our FRET values.
416
417 % ---- %
418
419 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
420 %% Filtering:
421
422 % Filter the data:
423 [DATA, REMOVEBURSTS] = FilterData(ArrivalMatrix, FILTERS); % this is a ...
    function!!!!
424
425
426
427 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
428 %% Burst Variance Analysis:
429
430 % Start the timer...
431 tic
432
433 % Launch the waitbar:
434 h.Waitbar = waitbar(...
435     0,...
436     'Commencing BVA...please be patient');
437
438 % ----- %
439 %% Winnow the Data:
440
441 % Replace all the NaNs:
442 ArrivalMatrix(isnan(ArrivalMatrix(:,3)),3) = 0;
443
444 % Return the assigned IDs of the various runs: % recall that non-burst
445 % photons are labeled with a '0'!!!!
446 FLAGVALS = SplitVec(...
447     ArrivalMatrix(:,3),...ArrivalMatrix(~isnan(ArrivalMatrix(:,3)),3),... ..
    % exclude any NaNs
448     'equal',... % find all those runs of the same index (burst ID)
449     'firstval'); % return just the first index marking a run.
450
451 % Extract some information about the bursts from the ArrivalMatrix:

```

```

452 STARTINDICES = SplitVec(...
453     ArrivalMatrix(:,3),... % exclude any NaNs
454     'equal',... % find all those runs of the same index (burst ID)
455     'first'); % return just the first index marking a run.
456
457 ENDINDICES = SplitVec(...
458     ArrivalMatrix(:,3),... % exclude any NaNs
459     'equal',... % find all those runs of the same index (burst ID)
460     'last'); % return just the last index completing a run.
461
462 STARTINDICES(FLAGVALS == 0) = []; % NOTE:
463 ENDINDICES(FLAGVALS == 0) = [];
464
465 % Get the start times and end times for each burst:
466 STARTTIMES = ArrivalMatrix(STARTINDICES, 1);
467 ENDTIMES = ArrivalMatrix(ENDINDICES, 1);
468
469 % Grab the total number of bursts:
470 TOTALBURSTS = numel(STARTINDICES);
471
472 % Create an index array:
473 BURSTINDICES(:,1) = 1 : TOTALBURSTS; % column vector
474
475 % Check!!!! Before binning photons, need to ensure that no StartTimes
476 % and EndTimes are equal (i.e., no burst ends when another begins), as
477 % this creates problems when creating the edges (eliminates a non-burst
478 % bin).
479 AdjacentBursts = intersect_sorted(...
480     STARTTIMES,...
481     ENDTIMES);
482
483 if isempty(AdjacentBursts) == false
484     % Adjacent bursts have been found!!!
485     fprintf('%d bursts were found to be immediately adjacent.\n',...
486         numel(AdjacentBursts));
487
488     % Cycle through each Adjacent Burst:
489     for ADJBURST = 1 : numel(AdjacentBursts)
490
491         % Find the burst:
492         AdjBurstIndex = find(...
493             STARTTIMES == AdjacentBursts(ADJBURST),...
494             1,...
495             'first');
496
497         % Increase the arrival time of any photons which
498         % arrived at the previous start time by a fractional amount:
499         ArrivalMatrix(ArrivalMatrix(:,1) == STARTTIMES(AdjBurstIndex, 1), ...
500             1) =...

```

```

500         ArrivalMatrix(ArrivalMatrix(:,1) == STARTTIMES(AdjBurstIndex, ...
501             1), 1)...
502         +...
503         1e-10; % should be less than NI card time resolution...
504
505         % Increase the start time of the next burst by a fractional amount:
506         STARTTIMES(AdjBurstIndex, 1) =...
507         STARTTIMES(AdjBurstIndex, 1)...
508         +...
509         1e-10;
510     end % end FOR
511
512
513 end % end IF
514
515
516 % Create Edges:
517 Edges = union_sorted_rows(...
518     STARTTIMES(:,1),...
519     ENDTIMES(:,1) + 1e-10);
520
521 % Histogram the arrival times:
522 [Counts, BinIndices] = histc(ArrivalMatrix(:,1), Edges);
523
524 % Determine which bins to throw away: (e.g., all those bins containing
525 % photons which do not belong to a burst)
526
527 % Discard the even BinIndices, as these correspond to bins defined by
528 % EdgeA = EndTime(1); EdgeB = StartTime(2). Hence, this bin (EdgeA,
529 % EdgeB) corresponds to photons which do not belong to a burst (i.e., photons
530 % occurring after the end of the previous burst and before the start of
531 % the next burst):
532 VALS1 = 0:2:max(BinIndices); % Note: we include 0 to account for photons ...
533     not falling into any defined bin.
534 TF = ismember_sorted(BinIndices, VALS1);
535
536 % Remove all photons not belonging to a burst, thus creating a 'burst
537 % arrival matrix':
538 ArrivalMatrix(TF == true,:) = [];
539 BinIndices(TF == true) = [];
540
541
542 % ----- %
543 %% Burst Removal:
544
545 % Remove the filtered bursts:
546 TF = ismember_sorted((BinIndices + 1) ./ 2, REMOVEBURSTS);

```

```

547
548 ArrivalMatrix(TF == true, :) = [];
549 BinIndices(TF == true, :) = [];
550
551
552 % Now, we need to rid ourselves of all photons originating from red
553 % excitation: DexDem = 0; DexAem = 2; AexAem = 1; AexDem = 3; => all
554 % photons labeled 1 or 3... (NOTE: this also removes acceptor-only
555 % bursts!)
556 LogicalIndexArray = (ArrivalMatrix(:,2) == 1 | ArrivalMatrix(:,2) == 3);
557 VALS2 = unique(BinIndices(LogicalIndexArray, :)); % This gives all burst ...
    indices which contain acceptor excitation photons
558
559 ArrivalMatrix(LogicalIndexArray,:) = []; % All acceptor excitation photons ...
    removed.
560 BinIndices(LogicalIndexArray,:) = []; % Note: this array will be important ...
    now, as all photons can be grouped by their unique identification with ...
    a Bin...
561 % NOTE: now VALS and unique(BinIndices) do not necessarily
562 % correspond!!!
563
564 % Determine which, if any, bursts have been removed from further
565 % analysis:
566 REMOVEBURSTS = [REMOVEBURSTS; (setdiff_sorted(VALS2, unique(BinIndices)) + ...
    1) ./ 2];
567
568 % REMOVEBURSTS now contains all bursts which either did not meet the
569 % filtering criteria above or were acceptor-only...
570
571 % Frequently, bursts have values which are not numbers; find them:
572 NaNBURSTS = find(isnan(DATA.Efficiency(:,1)));
573
574 REMOVEBURSTS = [REMOVEBURSTS; NaNBURSTS];
575
576
577
578
579
580 % ----- %
581 % Clustering: cluster the bursts according to mean Efficiency values:
582
583 % Define the cluster bounds: % NOTE: this creates an extra bin!!!
584 CLUSTEREDGES = -CLUSTERWIDTH/2:CLUSTERWIDTH:1+CLUSTERWIDTH/2;
585 CLUSTEREDGES = CLUSTEREDGES';
586
587 % Each burst is histogrammed into a cluster bin:
588 [Counts, CLUSTERBINIDX] = histc(...
589     DATA.Efficiency(:,1), ...
590     CLUSTEREDGES);

```



```

591
592
593
594 % ----- %
595
596 % Split the Arrival Matrix into Bursts:
597 IndexCell = SplitVec(...
598     BinIndices,...
599     'equal',... % Group all elements which are equal
600     'loc'); % Return the locations of these elements in a cellular array
601
602 % Each cellular element of IndexCell corresponds to a burst; within each
603 % cell element, the individual photons are assigned a location
604 % corresponding to their index in the (filtered) ArrivalMatrix.
605
606
607 % ----- %
608 %% BVA:
609
610 waitbar(0,...
611     h.Waitbar,...
612     'Performing sliding window analysis...');
613
614 % Initialize a BurstVarianceAnalysis array:
615 DATA.Results = nan(TOTALBURSTS, 2);
616
617 % Loop through the bursts and perform BVA:
618 COUNTER = 0;
619 CLUSTERSTATS = cell(CLUSTERBINS+1, 1);
620
621 TRACKER = 0;
622 for BURST = 1 : TOTALBURSTS
623
624     % Check!!!
625     if ismember(BURST, REMOVEBURSTS) % Determine if BURST is blacklisted...
626         % Blacklisted burst found! Move along to next burst...
627         continue;
628     else
629         % Update our counter for our non-blacklisted bursts...
630         COUNTER = COUNTER + 1;
631     end % end IF
632
633     % Grab the current (non-blacklisted) BURST indices:
634     INDICES = IndexCell{COUNTER};
635
636     % Remove a 'cutoff' number of photons:
637     if numel(INDICES) < PHOTONCUTOFF
638         % Go to next iteration...
639         REMOVEBURSTS(end+1,1) = BURST;

```

```

640     TRACKER = TRACKER + 1;
641     continue;
642 else
643     INDICES(1:PHOTONCUTOFF) = [];
644 end % end IF/ELSE
645
646
647 % Calculate total number of photons:
648 TOTALPHOTONS = numel(INDICES);
649
650 % Check!!!
651 if TOTALPHOTONS < WINDOW
652     % Go to next iteration, as not enough photons in 'burst'...
653     REMOVEBURSTS(end+1,1) = BURST;
654     TRACKER = TRACKER + 1;
655     continue;
656 end % end IF
657
658
659
660 % We must apply a window to the burst photons; this window is
661 % non-overlapping, and thus, in most cases, the number of windows
662 % within a burst is not an integer, but rather, a remainder of
663 % photons cannot be used. Let us discard these photons, so we can
664 % reshape our photon array:
665 RemPhotons = rem(TOTALPHOTONS, WINDOW);
666
667 % We have some leftover photons; remove them...
668 INDICES(end-RemPhotons+1:end) = [];
669
670
671 % Reshape the photons from the arrival matrix into an array of
672 % non-overlapping windows: (NOTE: photons within each window are
673 % placed along the columns, and the window number within the burst
674 % is placed along the rows)
675 PhotonArray = transpose(reshape(ArrivalMatrix(INDICES,2), WINDOW, ...
676     [])); % No apriori knowledge of how many windows within a burst
677
678 % Calculate the Efficiency value (FRET) within each window: (NOTE:
679 % we exploit the fact that DexDem photons are labeled 0):
680 EfficiencyVector = sum((PhotonArray == 2), 2) ./ WINDOW; % where ...
681     WINDOW = sum of DexDem + DexAem and the numerator is the number of ...
682     DexAem photons
683
684 % Calculate the standard deviation of the efficiency values:
685 EfficiencyStDev = std(EfficiencyVector);
686
687 % Place the mean and stdev of the efficiency in our BVA array:
688 DATA.Results(BURST, :) = [mean(EfficiencyVector), EfficiencyStDev];

```

```

686
687
688 % Place the efficiency vector with the burst's respective cluster:
689 WhichCluster = CLUSTERBINIDX(BURST);
690
691 % Check!!!
692 if WhichCluster == (CLUSTERBINS+2)
693     % This accounts for values which are precisely equal to the
694     % last edge. We place this data in the last bin.
695     WhichCluster = WhichCluster - 1;
696 end % end IF/ELSE
697
698 CLUSTERSTATS{WhichCluster} = [CLUSTERSTATS{WhichCluster}; ...
699     EfficiencyVector];
700
701 % Update our waitbar:
702 waitbar(BURST/TOTALBURSTS, h.Waitbar);
703
704
705 end % end FOR
706
707 fprintf(['%d bursts did not have sufficient photons ',...
708     'to perform a sliding window calculation.\n\n'],...
709     TRACKER);
710
711 % Tidy-up our Cluster Statistics:
712 RemoveClusters = cellfun(@isempty, CLUSTERSTATS);
713 CLUSTERSTATS(RemoveClusters) = [];
714 CLUSTEREDGES(RemoveClusters) = [];
715
716 RemoveClusters = cell2mat(cellfun(@(x) (numel(x)<CLUSTERMINWINDOWS),...
717     CLUSTERSTATS,...
718     'UniformOutput', false));
719 CLUSTERSTATS(RemoveClusters) = [];
720 CLUSTEREDGES(RemoveClusters) = [];
721
722
723 IDs = setdiff(BURSTINDICES, REMOVEBURSTS); % Retain only 'good' bursts
724
725 DATA.Results = DATA.Results(IDs,:);
726
727 % Remove any data in which the mean efficiency is either 0 or 1: (why?
728 % these 'events' tend to arise when having low photon statistics and will
729 % bunch up at the corner of the plots. Because of discrete photon
730 % statistics, these will occur often and affect the relatively weighting of
731 % the data elsewhere which has higher photon statistics and is more
732 % interesting (usually). Hence, we throw away this data, if present, and
733 % only keep that data which has more 'realistic' mean values.)

```

```

734 DATA.Results(DATA.Results(:,1) == 0    DATA.Results(:,1) == 1, :) = [];
735
736 % Remove any data in which the standard deviation is zero: (why? Having a
737 % single STD is not realistic, as fluctuations should be present simply to
738 % to photon statistics. This also might be indicative of insufficient
739 % number of WINDOWS allowing for STD calculation. Once again, with low
740 % photon statistics, this is possible, and can bias the weighting of your
741 % 2D histogram.)
742 DATA.Results(DATA.Results(:,2) == 0, :) = [];
743
744 % Remove all NaN rows:
745 DATA.Results(isnan(DATA.Results(:,1)), :) = [];
746
747
748 % Tidy-up:
749 clear ArrivalMatrix
750 clear BinIndices
751
752 % ----- %
753 %% Cluster Statistics:
754
755 waitbar(0,...
756     h.Waitbar,...
757     'Analyzing cluster statistics...');
758
759
760 % ----- %
761 % [ ] Group bursts with similar mean values and calculate population
762 % standard deviation:
763 ClusterCenters = CLUSTEREDGES(1:end-1) + CLUSTERWIDTH/2;
764
765 % Initialize some variables:
766 NumClusters = numel(CLUSTERSTATS);
767 HighCIvec = zeros(NumClusters, 1);
768 LowCIvec = zeros(NumClusters, 1);
769 StDevDistrMean = zeros(NumClusters, 1);
770
771 ClusterStats = nan(NumClusters, 2);
772
773 % Loop through each 'slice' (or 'cluster' of data) from the 2D Histogram
774 % and calculate the statistics:
775 for CLUSTER = 1 : NumClusters
776
777     ClusterStats(CLUSTER,2) = std(CLUSTERSTATS{CLUSTER});
778     ClusterStats(CLUSTER,1) = mean(CLUSTERSTATS{CLUSTER});
779
780     % Calculate StDev Distribution:
781     Edges = 0 : (1-0)/1000 : 0.6; % why 0.6? Vertical upper bound (could ...
        be something else, but for practical purposes, we make this realistic)

```

```

782 NumWindows = numel(CLUSTERSTATS{CLUSTER});
783 StDevDistr = MonteCarloPrediction(...
784     Edges,...
785     NumWindows,... % number of windows
786     WINDOW,... % number of photons within a window
787     ClusterCenters(CLUSTER),... % probability of energy transfer
788     NUMMONTECARLOSAMPLES); % number of samples generated by Monte ...
        Carlo simulation
789
790 CumDistr = cumsum(StDevDistr);
791 INV = 1 - CumDistr;
792 StDevDistrMean(CLUSTER) = sum(StDevDistr .* Edges); % What is this? ...
        This should correspond to the theoretical shot noise prediction ...
        curve based on the number of photons specified by WINDOW
793
794 %-- CI --
795 %-- ADJUST CI BY THE MULT HYPOTH TEST --
796 absvec = abs(CumDistr - CONFIDENCEINTERVAL);
797 [M, closest] = min(absvec);
798
799 absvecLOW = abs(INV - CONFIDENCEINTERVAL);
800 closestLOW = find(absvecLOW == min(absvecLOW),...
801     1, 'last');
802
803 % Why are we calculating the CI off an index? The index tells us
804 % something of about the number of false positives.
805 HighCIvec(CLUSTER,:) = (closest-1)*.001; % 0.001 is 99.9% Confidence ...
        interval
806 LowCIvec(CLUSTER,:) = (closestLOW-1)*.001;
807
808 % Update our waitbar:
809 waitbar(CLUSTER / NumClusters,...
810     h.Waitbar);
811
812
813 end % end FOR
814
815 % Assign over to application data:
816 DATA.Clusters.Centers = ClusterCenters;
817 DATA.ConfidenceInterval.Upper = HighCIvec;
818 DATA.ConfidenceInterval.Lower = LowCIvec;
819 DATA.Clusters.StDevDistrMean = StDevDistrMean;
820 DATA.Clusters.Statistics = ClusterStats;
821 DATA.Clusters.WindowMeans = CLUSTERSTATS;
822 DATA.Clusters.DistanceFromUpperCI =...
823     ClusterStats(:,2) - HighCIvec;
824
825
826 % ----- %

```

```
827 %% Tidy-up:
828
829
830
831 % Delete waitbar:
832 delete(h.Waitbar);
833
834
835 TOTALTIME = toc
836
837
838
839
840
841
842 return;
843 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
844 %% EOF
845
846
847
848 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
849 %% StartUp:
850 function StartUp()
851 %
852 %
853 %
854
855 addpath(...
856     'Other/lightspeed',...
857     'Other/SplitVec');
858
859
860
861 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
862 %% CleanUp:
863 function CleanUp()
864 %
865 %
866 %
867
868 rmpath(...
869     'Other/lightspeed',...
870     'Other/SplitVec');
```

Data Filtering

```

1 function [DATA, REMOVEBURSTS] = FilterData(...)
2     ArrivalMatrix, FILTERS)
3 % FILTERDATA
4 %     Filter a photon arrival matrix using supplied filter parameters.
5 %
6 %     INPUTS:
7 %         ARRIVALMATRIX: input data on which to perform Burst Variance
8 %         Analysis. The input matrix should be Nx3 or 3xN, with the following
9 %         as vector inputs: (1) Photon Arrival Times; (2) Detection Channel
10 %        (DexDem = 0; DexAem = 2; AexAem = 1; AexDem = 3); (3) Burst
11 %        Classification (those photons belonging to a burst should have a
12 %        corresponding index to a unique burst ID; background photons should
13 %        be indexed as NaN): ARRIVALMATRIX = ArrivalTimes    ID    BurstNumber
14 %
15 %        FILTERS: nested structure providing minimum and maximum thresholds
16 %        for the following data streams:
17 %            DURATION: the temporal length of a detected burst [seconds]
18 %            LENGTH: the number of photons within a detected burst [photons]
19 %            DEXDEM: the number of photons detected in the donor channel
20 %                upon donor excitation.
21 %            DEXAEM: the number of photons detected in the acceptor channel
22 %                upon donor excitation.
23 %            AEXAEM: the number of photons detected in the acceptor channel
24 %                upon acceptor excitation.
25 %            SUMPHOTONS: the number of photons detected in the donor &
26 %                acceptor channels upon donor excitation.
27 %            TOTALPHOTONS: the number of photons detected in the donor &
28 %                acceptor channels upon donor and acceptor excitation.
29 %                (excluding photons detected in donor channel upon acceptor
30 %                excitation: AexDem)
31 %            EFFICIENCY: the FRET efficiency.
32 %            STOICHIOMETRY: the relative fluorophore brightness.
33 %
34 %     OUTPUTS:
35 %         DATA: structure with the following data streams for each detected
36 %         burst:
37 %             DURATION: the temporal length of a detected burst [seconds]
38 %             LENGTH: the number of photons within a detected burst [photons]
39 %             DEXDEM: the number of photons detected in the donor channel
40 %                 upon donor excitation.
41 %             DEXAEM: the number of photons detected in the acceptor channel
42 %                 upon donor excitation.
43 %             AEXAEM: the number of photons detected in the acceptor channel
44 %                 upon acceptor excitation.
45 %             SUMPHOTONS: the number of photons detected in the donor &

```

```

46 %           acceptor channels upon donor excitation.
47 %           TOTALPHOTONS: the number of photons detected in the donor &
48 %           acceptor channels upon donor and acceptor excitation.
49 %           (excluding photons detected in donor channel upon acceptor
50 %           excitation: AexDem)
51 %           EFFICIENCY: the FRET efficiency.
52 %           STOICHIOMETRY: the relative fluorophore brightness.
53 %
54 %           REMOVEBURSTS: a vector providing indices for the bursts which
55 %           failed filter criteria.
56 %
57 %   DEPENDENCIES:
58 %       SPLITVEC.M
59 %
60 %   -----
61 %   EXAMPLE USAGE:
62 %       [DATA, REMOVEBURSTS] = FilterData(ArrivalMatrix, FILTERS);
63 %
64 %   -----
65 %   History:
66 %       [1] Kristofer Gryte. University of Oxford (2011).
67 %
68 %   Copyright (c) 2011. Kristofer Gryte. University of Oxford.
69 %
70 %
71 %
72 %   Version: 1.0 (2011-07-30)
73 %
74 %   -----
75 %   Author Information:
76 %       Kristofer Gryte
77 %       Clarendon Laboratory
78 %       University of Oxford
79 %       Oxford, United Kingdom
80 %       e-mail: k.gryte1 (at) physics.ox.ac.uk
81 %
82 %   -----
83 %   TODO:
84 %       [1] Add checks for input variables.
85 %
86 %
87 %
88 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89 %% EOP
90
91
92 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
93 %% Initialization:
94

```



```

95 REMOVEBURSTS = [];
96
97 Apply = true; % FLAG to determine if apply filters or not. THIS IS NEEDED, ...
    FOR NOW, EVEN IF NO FILTERS ARE APPLIED!!!!
98
99 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
100 %% Assemble Single Data Matrix:
101
102 % Replace all the NaNs:
103 ArrivalMatrix(isnan(ArrivalMatrix(:,3)),3) = 0;
104
105 % Return the assigned IDs of the various runs: % recall that non-burst
106 % photons are labeled with a '0'!!!!
107 FLAGVALS = SplitVec(...
108     ArrivalMatrix(:,3),...ArrivalMatrix(~isnan(ArrivalMatrix(:,3)),3),... ...
    % exclude any NaNs
109     'equal',... % find all those runs of the same index (burst ID)
110     'firstval'); % return just the first index marking a run.
111
112
113 % --- %
114 % [1] Start Times:
115 STARTINDICES = SplitVec(...
116     ArrivalMatrix(:,3),...ArrivalMatrix(~isnan(ArrivalMatrix(:,3)),3),... ...
    % exclude any NaNs
117     'equal',... % find all those runs of the same index (burst ID)
118     'first'); % return just the first index marking a run.
119
120 STARTINDICES(FLAGVALS == 0) = []; % NOTE:
121
122 DATA.StartTimes = ArrivalMatrix(STARTINDICES, 1);
123
124 % --- %
125 % [2] End Times:
126 ENDINDICES = SplitVec(...
127     ArrivalMatrix(:,3),...ArrivalMatrix(~isnan(ArrivalMatrix(:,3)),3),... ...
    % exclude any NaNs
128     'equal',... % find all those runs of the same index (burst ID)
129     'last'); % return just the last index completing a run.
130
131 ENDINDICES(FLAGVALS == 0) = []; % NOTE:
132
133 DATA.EndTimes = ArrivalMatrix(ENDINDICES, 1);
134
135 % --- %
136 % [3] Duration:
137 DATA.Duration = DATA.EndTimes - DATA.StartTimes;
138
139 % --- %

```

```

140 % [4] Length:
141 DATA.Length = ENDINDICES - STARTINDICES;
142
143 % --- %
144 BURSTCELL(:,1) = SplitVec(...
145     ArrivalMatrix(:, 3),...ArrivalMatrix(~isnan(ArrivalMatrix(:,3)), 2),...
146     'equal',...
147     'loc'); % this is a cellular array!
148
149 BURSTCELL(FLAGVALS == 0) = []; % NOTE:
150
151 % ----- %
152 % Free up some memory:
153 % clear ArrivalMatrix
154 % ----- %
155
156 % [5] DexDem: % DexDem = 0
157 DATA.DexDem(:,1) = cellfun(@(x) sum(ArrivalMatrix(x,2) == 0), BURSTCELL, ...
158     'UniformOutput', true);
159
160 % [6] DexAem: % DexAem = 2
161 DATA.DexAem(:,1) = cellfun(@(x) sum(ArrivalMatrix(x,2) == 2), BURSTCELL, ...
162     'UniformOutput', true);
163
164 % [7] AexAem: % AexAem = 1
165 DATA.AexAem(:,1) = cellfun(@(x) sum(ArrivalMatrix(x,2) == 1), BURSTCELL, ...
166     'UniformOutput', true);
167
168 % --- %
169 % [8] Sum Photons: DexDem + DexAem
170 DATA.SumPhotons = DATA.DexDem + DATA.DexAem;
171
172 % --- %
173 % [9] Total Photons: DexDem + DexAem + AexAem
174 DATA.TotalPhotons = DATA.SumPhotons + DATA.AexAem;
175
176 % --- %
177 % [10] Efficiency:
178 DATA.Efficiency = DATA.DexAem ./ (DATA.SumPhotons);
179
180 % --- %
181 % [11] Stoichiometry:
182 DATA.Stoichiometry = DATA.SumPhotons ./ (DATA.TotalPhotons);
183
184 % Threshold Data:
185 if Apply == true

```

```

186
187 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
188 % Min Duration:
189 if ~isempty(FILTERS.Duration.Min)
190
191     IDs = find(DATA.Duration < FILTERS.Duration.Min); % Column vector
192     REMOVEBURSTS = [REMOVEBURSTS; IDs];
193
194     DATA.Duration(DATA.Duration < FILTERS.Duration.Min,:) = NaN;
195
196 end
197
198 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
199 % Max Duration:
200 if ~isempty(FILTERS.Duration.Max)
201
202     IDs = find(DATA.Duration > FILTERS.Duration.Max); % Column vector
203     REMOVEBURSTS = [REMOVEBURSTS; IDs];
204
205     DATA.Duration(DATA.Duration > FILTERS.Duration.Max,:) = NaN;
206
207 end
208
209
210 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
211 % Min Length:
212 if ~isempty(FILTERS.Length.Min)
213
214     IDs = find(DATA.Length < FILTERS.Length.Min); % Column vector
215     REMOVEBURSTS = [REMOVEBURSTS; IDs];
216
217     DATA.Length(DATA.Length < FILTERS.Length.Min,:) = NaN;
218
219 end
220
221 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
222 % Max Length:
223 if ~isempty(FILTERS.Length.Max)
224
225     IDs = find(DATA.Length > FILTERS.Length.Max); % Column vector
226     REMOVEBURSTS = [REMOVEBURSTS; IDs];
227
228     DATA.Length(DATA.Length > FILTERS.Length.Max,:) = NaN;
229
230 end
231
232
233
234 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

235 % Min DexcDem:
236 if ~isempty(FILTERS.DexDem.Min)
237
238     IDs = find(DATA.DexDem < FILTERS.DexDem.Min); % Column vector
239     REMOVEBURSTS = [REMOVEBURSTS; IDs];
240
241     DATA.DexDem(DATA.DexDem < FILTERS.DexDem.Min,:) = NaN;
242
243 end
244
245 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
246 % Max DexcDem:
247 if ~isempty(FILTERS.DexDem.Max)
248
249     IDs = find(DATA.DexDem > FILTERS.DexDem.Max); % Column vector
250     REMOVEBURSTS = [REMOVEBURSTS; IDs];
251
252     DATA.DexDem(DATA.DexDem > FILTERS.DexDem.Max,:) = NaN;
253
254 end
255
256 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
257 % Min DexcAem:
258 if ~isempty(FILTERS.DexAem.Min)
259
260     IDs = find(DATA.DexAem < FILTERS.DexAem.Min); % Column vector
261     REMOVEBURSTS = [REMOVEBURSTS; IDs];
262
263     DATA.DexAem(DATA.DexAem < FILTERS.DexAem.Min,:) = NaN;
264
265 end
266
267 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
268 % Max DexcAem:
269 if ~isempty(FILTERS.DexAem.Max)
270
271     IDs = find(DATA.DexAem > FILTERS.DexAem.Max); % Column vector
272     REMOVEBURSTS = [REMOVEBURSTS; IDs];
273
274     DATA.DexAem(DATA.DexAem > FILTERS.DexAem.Max,:) = NaN;
275
276 end
277
278 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
279 % Min AexcAem:
280 if ~isempty(FILTERS.AexAem.Min)
281
282     IDs = find(DATA.AexAem < FILTERS.AexAem.Min); % Column vector
283     REMOVEBURSTS = [REMOVEBURSTS; IDs];

```

```

284
285     DATA.AexAem(DATA.AexAem < FILTERS.AexAem.Min,:) = NaN;
286
287 end
288
289 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
290 % Max AexcAem:
291 if ~isempty(FILTERS.AexAem.Max)
292
293     IDs = find(DATA.AexAem > FILTERS.AexAem.Max); % Column vector
294     REMOVEBURSTS = [REMOVEBURSTS; IDs];
295
296     DATA.AexAem(DATA.AexAem > FILTERS.AexAem.Max,:) = NaN;
297
298 end
299
300 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
301 % Min Sum DD+DA:
302 if ~isempty(FILTERS.SumPhotons.Min)
303
304     IDs = find(DATA.SumPhotons < FILTERS.SumPhotons.Min); % Column vector
305     REMOVEBURSTS = [REMOVEBURSTS; IDs];
306
307     DATA.SumPhotons(DATA.SumPhotons < FILTERS.SumPhotons.Min,:) = NaN;
308
309 end
310
311 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
312 % Max Sum DD+DA:
313 if ~isempty(FILTERS.SumPhotons.Max)
314
315     IDs = find(DATA.SumPhotons > FILTERS.SumPhotons.Max); % Column vector
316     REMOVEBURSTS = [REMOVEBURSTS; IDs];
317
318     DATA.SumPhotons(DATA.SumPhotons > FILTERS.SumPhotons.Max,:) = NaN;
319
320 end
321
322 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
323 % Min Total Signal:
324 if ~isempty(FILTERS.TotalPhotons.Min)
325
326     IDs = find(DATA.TotalPhotons < FILTERS.TotalPhotons.Min); % Column ...
327     REMOVEBURSTS = [REMOVEBURSTS; IDs];
328
329     DATA.TotalPhotons(DATA.TotalPhotons < FILTERS.TotalPhotons.Min,:) ...
330     = NaN;

```

```

331 end
332
333 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
334 % Max Total Signal:
335 if ~isempty(FILTERS.TotalPhotons.Max)
336
337     IDs = find(DATA.TotalPhotons > FILTERS.TotalPhotons.Max); % Column ...
338         vector
339     REMOVEBURSTS = [REMOVEBURSTS; IDs];
340
341     DATA.TotalPhotons(DATA.TotalPhotons > FILTERS.TotalPhotons.Max,:) ...
342         = NaN;
343
344 end
345
346 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
347 % Min Efficiency:
348 if ~isempty(FILTERS.Efficiency.Min)
349
350     IDs = find(DATA.Efficiency < FILTERS.Efficiency.Min); % Column vector
351     REMOVEBURSTS = [REMOVEBURSTS; IDs];
352
353     DATA.Efficiency(DATA.Efficiency < FILTERS.Efficiency.Min,:) = NaN;
354
355 end
356
357 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
358 % Max Efficiency:
359 if ~isempty(FILTERS.Efficiency.Max)
360
361     IDs = find(DATA.Efficiency > FILTERS.Efficiency.Max); % Column vector
362     REMOVEBURSTS = [REMOVEBURSTS; IDs];
363
364     DATA.Efficiency(DATA.Efficiency > FILTERS.Efficiency.Max,:) = NaN;
365
366 end
367
368 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
369 % Min Stoichiometry:
370 if ~isempty(FILTERS.Stoichiometry.Min)
371
372     IDs = find(DATA.Stoichiometry < FILTERS.Stoichiometry.Min); % ...
373         Column vector
374     REMOVEBURSTS = [REMOVEBURSTS; IDs];
375
376     DATA.Stoichiometry(DATA.Stoichiometry < ...
377         FILTERS.Stoichiometry.Min,:) = NaN;
378
379 end

```

```
376
377 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
378 % Max Stoichiometry:
379 if ~isempty(FILTERS.Stoichiometry.Max)
380
381     IDs = find(DATA.Stoichiometry > FILTERS.Stoichiometry.Max); % ...
382     Column vector
383     REMOVEBURSTS = [REMOVEBURSTS; IDs];
384
385     DATA.Stoichiometry(DATA.Stoichiometry > ...
386         FILTERS.Stoichiometry.Max,:) = NaN;
387
388 end
389
390 end % end IF
391
392 % Retain only the unique indices:
393 REMOVEBURSTS = unique(REMOVEBURSTS);
394
395
396
397 % FOR NOW...
398 return;
399
400
401
402 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
403 %% EOF
```

Monte Carlo Simulation

```

1 function STDDISTRIBUTION = MonteCarloPrediction(EDGES, EVENTS,...
2     TRIALS, PROBABILITY, ITERATIONS)
3 % MONTECARLOPREDICTION
4 % Use Monte Carlo methods to generate a distribution of standard
5 % deviation (std) values observed from a binomial process (Bernoulli
6 % Trial) with a specified probability of success.
7 %
8 % INPUTS:
9 %     EDGES: a vector defining the bin edges into which the simulated
10 % standard deviations will be histogrammed.
11 %
12 %     EVENTS: the number of events (collections of trials) to be
13 % sampled from the distribution.
14 %
15 %     TRIALS: the number of trials.
16 %
17 %     PROBABILITY: the probability of success. For example, a series of
18 % coin flips is a collection of events characterized by Bernoulli
19 % trials with probability 0.5.
20 %
21 %     ITERATIONS: the number of times to repeat each collection of events
22 % so as to arrive at an average value for the calculated standard
23 % deviations. Default: 500.
24 %
25 %
26 % OUTPUTS:
27 %     STDDISTRIBUTION: a probability distribution of the standard
28 % deviation with support defined by EDGES. Output is a vector.
29 %
30 % DEPENDENCIES:
31 %     BINORND.M (STATS toolbox)
32 %
33 % -----
34 % EXAMPLE USAGE:
35 %     MonteCarloPrediction([0:0.0001:1], 5, 100, 0.5);
36 %
37 %     MonteCarloPrediction([0:0.0001:1], 5, 100, 0.5, 500);
38 %
39 % -----
40 % History:
41 %     [1] Joseph Torella, Yusdi Santoso. University of Oxford (2009).
42 %     [2] Kristofer Gryte. University of Oxford (2011).
43 %
44 % Copyright (c) 2011. Kristofer Gryte. University of Oxford.
45 %

```



```

46 %
47 %
48 % Version: 1.0 (2011-07-30)
49 %
50 % -----
51 % Author Information:
52 %     Kristofer Gryte
53 %     Clarendon Laboratory
54 %     University of Oxford
55 %     Oxford, United Kingdom
56 %     e-mail: k.gryte1 (at) physics.ox.ac.uk
57 %
58 % -----
59 % TODO:
60 %     [1] Generate a function independent of the STATS toolbox.
61 %
62 %
63 % -----
64 % See also BINORND
65 %
66 %
67
68 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69 %% EOP
70
71
72
73
74
75 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76 %% Check Input Arguments:
77
78 % Check!
79 if nargin ≤ 4
80     ITERATIONS = 500; % default value
81 end % end IF
82
83 % Check EDGES:
84 if size(EDGES,1) ≠ 1 && size(EDGES,2) ≠ 1
85     % Throw an error:
86     errordlg(['ERROR:invalid input argument. Please input a vector for the ...
87             ',...
88             'EDGES argument.'],...
89             'ERROR:invalid input argument');
89
90     return;
91 end % end IF
92
93 % Check EVENTS, TRIALS, PROBABILITY, and ITERATIONS

```

```

94 temp = [EVENTS, TRIALS, ITERATIONS];
95 if sum((~isinteger(temp)) + (temp>0)) ≠ 6
96     % Throw an error:
97     errordlg(['ERROR:invalid input argument. Please input positive ...
98             integers for the ',...
99             'EVENTS, TRIALS, and ITERATIONS arguments.'],...
100             'ERROR:invalid input argument');
101
102     return;
103 end % end IF
104
105 if (PROBABILITY ≤ 0      PROBABILITY > 1)
106     % Throw an error:
107     errordlg(['ERROR:invalid input argument. Please input a probability ...
108             between 0 and 1 ',...
109             'for the PROBABILITY argument.'],...
110             'ERROR:invalid input argument');
111
112     return;
113 end % end IF
114
115
116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
117 %% Initialization:
118
119 % --- %
120 % Parameters:
121 Runs = 3; % used in the event of a large number of requested standard ...
122           % deviations, in which case, for memory purposes, we need to break the ...
123           % work into smaller chunks.
124
125 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
126 %% Generate Monte Carlo Prediction:
127
128 if EVENTS*ITERATIONS ≤ 10000 % 10,000 ? A magical number discovered ...
129     heuristically.
130
131     % Generate the matrix of numbers drawn from a binomial distribution
132     % with total trials equal to TRIALS and probability equal to
133     % PROBABILITY:
134     MyMatrix = binornd(...
135         TRIALS,...
136         PROBABILITY,...
137         EVENTS,...
138         ITERATIONS); % this is a matrix of size: EVENTS x ITERATIONS

```

```
138
139     % Calculate the standard deviation along each column
140     StdMatrix = std(MyMatrix ./ TRIALS);
141
142 elseif EVENTS*ITERATIONS > 10000
143
144     % With many iterations, we run into the problem of very large matrices
145     % and memory issues. Let us break it down more manageably:
146     % ---- %
147
148     RemainingSamples = ITERATIONS;
149
150     % Pre-allocate our array:
151     StdMatrix = nan(1, ITERATIONS);
152
153     % Initialize an index counter:
154     Counter = 1;
155
156     for k = 1 : Runs
157
158         % Determine the number of samples to run:
159         Samples = floor(RemainingSamples / (Runs-k+1));
160
161         % Generate the matrix of numbers drawn from a binomial distribution
162         % with total trials equal to TRIALS and probability equal to
163         % PROBABILITY:
164         MyMatrix = binornd(...
165             TRIALS,...
166             PROBABILITY,...
167             EVENTS,...
168             Samples);
169
170         % Calculate the standard deviation along each column and append to
171         % our standard deviation matrix:
172         StdMatrix(Counter : (Counter+Samples-1)) = std(MyMatrix ./ TRIALS);
173
174         % Calculate the remaining number of samples:
175         RemainingSamples = RemainingSamples - Samples;
176
177         % Update our counter:
178         Counter = Counter + Samples;
179
180
181     end % end FOR
182
183
184 end % end IF/ELSEIF
185
186 % Histogram the standard deviations where the bin edges are defined by
```

```
187 % EDGES:
188 [Counts, BinIndices] = histc(StdMatrix, EDGES);
189
190 % Determine the probability of a particular standard deviation, by
191 % normalizing by the total number of counts:
192 STDDISTRIBUTION = Counts / sum(Counts); % this is a vector
193
194
195 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
196 %% EOF
```

Visualization

```

1 function [DATA, handles] = PlotResults(DATA, WINDOW, handles)
2 % PLOTRESULTS
3 %   Plots the DATA results generated by Burst Variance Analysis into the
4 %   axes supplied by HANDLES. Configuration and aesthetics of the axes is
5 %   currently controlled internally. NOTE: the BVA data is smoothed from a
6 %   discretized histogram presentation and transformed to a continuous
7 %   contour. The reference for this smoothing is " Garcia D, Robust
8 %   smoothing of gridded data in one and higher dimensions with missing
9 %   values. Computational Statistics & Data Analysis, 2010. "
10 %
11 %   INPUTS:
12 %       DATA: structure with the following data streams for each detected
13 %       burst: (NOTE: data streams are filtered according to input FILTERS)
14 %           DURATION: the temporal length of a detected burst [seconds]
15 %           LENGTH: the number of photons within a detected burst [photons]
16 %           DEXDEM: the number of photons detected in the donor channel
17 %               upon donor excitation.
18 %           DEXAEM: the number of photons detected in the acceptor channel
19 %               upon donor excitation.
20 %           AEXAEM: the number of photons detected in the acceptor channel
21 %               upon acceptor excitation.
22 %           SUMPHOTONS: the number of photons detected in the donor &
23 %               acceptor channels upon donor excitation.
24 %           TOTALPHOTONS: the number of photons detected in the donor &
25 %               acceptor channels upon donor and acceptor excitation.
26 %               (excluding photons detected in donor channel upon acceptor
27 %               excitation: AexDem)
28 %           EFFICIENCY: the FRET efficiency.
29 %           STOICHIOMETRY: the relative fluorophore brightness.
30 %           RESULTS: an Nx2 array, where N is the number of bursts (after
31 %               filtering). The first column corresponds to the mean ...
32 %               Efficiency
33 %               value for a burst and the second to the burst's Efficiency
34 %               standard deviation. Array: E(efficiency)  sqrt(V(efficiency))
35 %           CLUSTERS: nested structure with the following fields:
36 %               CENTERS: vector containing the centers of the individual
37 %               clusters.
38 %               STDEVDIRMEAN: the Monte Carlo simulated standard
39 %               deviation prediction.
40 %               STATISTICS: an Mx2 array, where M is the number of clusters
41 %               ('slices' along the efficiency axis, into which bursts are
42 %               grouped). The first column corresponds to the mean
43 %               Efficiency value for the cluster, and the second to the
44 %               standard deviation of the burst's within that cluster.
45 %           CONFIDENCEINTERVAL: nested structure with the following fields:

```

```

45 %         UPPER: the upper confidence bound generated from the Monte
46 %             Carlo simulation and multiple hypothesis test.
47 %         LOWER: the lower confidence bound generated from the Monte
48 %             Carlo simulation and the multiple hypothesis test.
49 %
50 %     WINDOW: size of the sliding window over which FRET will be
51 %     calculated. Default: 5.
52 %
53 %     HANDLES: structure with the following fields:
54 %         FIGURE: handle to the figure window.
55 %         SUBPLOT: nested structure with the following fields:
56 %             BVA: nested structure with the following fields:
57 %                 AXES: handles to the 2D Histogram axes (subplot).
58 %         SUBPLOT: nested structure with the following fields:
59 %             HISTOGRAM: nested structure with the following fields:
60 %                 MEAN: nested structure with the following fields:
61 %                     AXES: handle to the axes into which the histogram
62 %                         will be plotted.
63 %                 STDEV: nested structure with the following fields:
64 %                     AXES: handle to the axes into which the histogram
65 %                         will be plotted.
66 %
67 %
68 %
69 %     OUTPUTS:
70 %         DATA: input structure with the following added data streams:
71 %             HISTOGRAMS: nested structure with the following fields:
72 %                 COUNTS: the counts per bin for the 2D histogram.
73 %                 EDGES: the edges used to define the histogram bins.
74 %                 BINCENTERS: the locations of the center of each bin.
75 %                 SMOOTHED: the histogram data after application of a
76 %                     smoothing filter.
77 %
78 %         HANDLES: input structure with the following added fields:
79 %             PLOTS: nested structure with the following fields:
80 %                 CONTOUR: handle to the counter plot created from the 2D
81 %                     histogram
82 %                 SHOTNOISEPREDICTION: handle to the shot noise prediction
83 %                     based on the number of photons specified in WINDOW.
84 %             CONFIDENCEINTERVAL: nested structure with the following
85 %                 fields:
86 %                 UPPER: handle to the upper confidence interval
87 %                     generated via a Monte Carlo simulation. Cluster STD
88 %                     values above this limit are indicative of dynamics.
89 %                 MEAN: handle to the average of upper and lower
90 %                     confidence bounds. This should correspond to the
91 %                     SHOTNOISEPREDICTION curve.
92 %             STATIC: STD clusters which fall below the upper confidence
93 %                 interval are considered 'static'. This is the handle to

```

```

94 %             these data points.
95 %         DYNAMICS: STD clusters which are above the upper confidence
96 %             interval are considered 'dynamic'. This is the handle to
97 %             these data points.
98 %         UNKNOWN: STD clusters which fall below the lower confidence
99 %             interval are categorized as 'unknown'. The causes of this
100 %            behavior are not well understood.
101 %         SUBPLOT: nested structure with the following fields:
102 %             HISTOGRAM: nested structure with the following fields:
103 %                 MEAN: nested structure with the following fields:
104 %                     PLOTS: nested structure with the following fields:
105 %                         BAR: handle to the Mean bar graph data.
106 %                 STDEV: nested structure with the following fields:
107 %                     PLOTS: nested structure with the following fields:
108 %                         BAR: handles to the StDev bar graph data.
109 %         COLORBAR: handle to the colobar corresponding to the 2D
110 %            histogram data.
111 %
112 %
113 %
114 %     DEPENDENCIES:
115 %         SMOOTHN.M
116 %         HISTCN.M
117 %
118 %     -----
119 %     EXAMPLE USAGE:
120 %         [DATA, handles] = PlotResults(DATA, WINDOW, handles);
121 %
122 %     -----
123 %     History:
124 %         [1] Kristofer Gryte. University of Oxford (2011).
125 %
126 %     Copyright (c) 2011. Kristofer Gryte. University of Oxford.
127 %
128 %
129 %
130 %     Version: 1.0 (2011-08-02)
131 %
132 %     -----
133 %     Author Information:
134 %         Kristofer Gryte
135 %         Clarendon Laboratory
136 %         University of Oxford
137 %         Oxford, United Kingdom
138 %         e-mail: k.grytel (at) physics.ox.ac.uk
139 %
140 %     -----
141 %     TODO:
142 %         [1] Allow axes configurations to be an input structure.

```

```

143 %           [2] Provide checks on input arguments.
144 %           [3] Figure out why 'histcn' appears to reset the path(s).
145 %
146 %
147
148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
149 %% EOP
150
151
152 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
153 %% StartUp/CleanUp:
154
155 StartUp();
156
157 C = onCleanup(@() Cleanup());
158
159
160 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
161 %% Initialization:
162
163 % --- %
164 % Check!!!
165 if nargin ≠ 3
166     % Throw an error:
167     errordlg(['ERROR:not enough input arguments. Please provide the ...
168             following ',...
169             'input arguments: DATA, WINDOW, HANDLES. See the HELP for the more ...
170             ',...
171             'information.'],...
172             'ERROR:not enough input arguments');
173     return;
174 end % end IF
175
176 % --- %
177 % Default Configurations:
178
179 Config.Histogram2D.Smoothing = 20; % Smoothing parameter
180
181 Config.ShotNoise.Prediction.OrdinateVals = 0:0.01:1; % Efficiency sampling....
182 Config.ShotNoise.Prediction.StDev = sqrt(...
183     (Config.ShotNoise.Prediction.OrdinateVals .* ...
184     (1-Config.ShotNoise.Prediction.OrdinateVals))...
185     /...
186     WINDOW);
187
188 Config.Axes.XLabel.FontSize = 14;
189 Config.Axes.XLabel.String = 'Mean';
190
191 Config.Axes.YLabel.FontSize = 14;

```



```

189 Config.Axes.YLabel.String = 'StDev';
190
191 Config.Axes.Title.FontSize = 14;
192 Config.Axes.Title.String = 'Burst Variance Analysis (Efficiency)';
193
194 Config.Axes.TickDir = 'out';
195 Config.Axes.Box = 'off';
196
197 Config.Figure.BackgroundColor = 'w';
198
199 Config.Legend.Box = 'off';
200 Config.Legend.Location = 'south';
201
202
203 Config.Histogram.Mean.Axes.YLabel.FontSize = 14;
204 Config.Histogram.Mean.Axes.YLabel.String = 'Counts';
205
206 Config.Histogram.Mean.Axes.Box = 'off';
207 Config.Histogram.Mean.Axes.TickDir = 'out';
208
209
210 Config.Histogram.StDev.Axes.XLabel.FontSize = 14;
211 Config.Histogram.StDev.Axes.XLabel.String = 'Counts';
212
213 Config.Histogram.StDev.Axes.Box = 'off';
214 Config.Histogram.StDev.Axes.TickDir = 'out';
215
216 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
217
218
219 % ----- %
220 %% Plot:
221
222 % Activate our figure:
223 figure(handles.figure);
224
225 % Activate our BVA Axes:
226 subplot(handles.Subplot.BVA.Axes);
227
228 % Get the histogram counts: (NOTE: default is 100 Bins)
229 [DATA.Histogram.Counts, DATA.Histogram.Edges, DATA.Histogram.BinCenters] = ...
    histcn(...
230     DATA.Results);
231
232 StartUp(); %% NOTE!!! For some reason, 'histc'/'histcn' appears to reset ...
    the path(s)
233
234 % 'Smooth' the histogram:
235 DATA.Histogram.Smoothed = smoothn(...

```

```

236     DATA.Histogram.Counts,...
237     Config.Histogram2D.Smoothing);
238
239 % ----- %
240 % [1] Create the contour plot of the burst standard deviations:
241 [ContourMatrix, handles.Plots.Contour] = contourf(...
242     DATA.Histogram.BinCenters{1},...
243     DATA.Histogram.BinCenters{2},...
244     DATA.Histogram.Smoothed(:, :, 1)',...
245     10,...
246     'EdgeColor', 'none',...
247     'tag', 'Per Burst StDev');
248
249 hold(handles.Subplot.BVA.Axes, 'on');
250
251 % ----- %
252 % [2] Display the shot noise prediction:
253 handles.Plots.ShotNoisePrediction = plot(...
254     Config.ShotNoise.Prediction.OrdinateVals,...
255     Config.ShotNoise.Prediction.StDev,...
256     'k-',...
257     'LineWidth', 3.0,...
258     'tag', 'Shot Noise Prediction');
259
260 % ----- %
261 % [3] Plot the confidence bounds:
262 handles.Plots.ConfidenceInterval.Upper = plot(...
263     DATA.Clusters.Centers,...
264     DATA.ConfidenceInterval.Upper,...
265     'm-',...
266     'LineWidth', 2.5,...
267     'tag', 'Upper CI');
268
269 handles.Plots.ConfidenceInterval.Mean = plot(...
270     DATA.Clusters.Centers(:, 1),...
271     DATA.Clusters.StDevDistrMean(:, 1),...
272     'y--',...
273     'LineWidth', 2.0,...
274     'MarkerSize', 10,...
275     'tag', 'Mean MC Prediction');
276
277 % ----- %
278 % [4] Determine where the clustered populations reside relative to the
279 % confidence bounds:
280 LogID = (DATA.Clusters.Statistics(:, 2) ≤ DATA.ConfidenceInterval.Upper...
281     ...
282     DATA.Clusters.Statistics(:, 2) ≥ DATA.ConfidenceInterval.Lower);
283
284 handles.Plots.Static = plot(...

```

```

285 DATA.Clusters.Centers(LogID, 1),...
286 DATA.Clusters.Statistics(LogID, 2),...
287 'k^',...
288 'LineWidth', 1.5,...
289 'MarkerFaceColor', 'k',...
290 'MarkerSize', 8,...
291 'tag', 'Static');
292
293
294 LogID = (DATA.Clusters.Statistics(:,2) > DATA.ConfidenceInterval.Upper);
295 handles.Plots.Dynamics = plot(...
296     DATA.Clusters.Centers(LogID, 1),...
297     DATA.Clusters.Statistics(LogID, 2),...
298     'k^',...
299     'LineWidth', 1.5,...
300     'MarkerFaceColor', 'r',...
301     'MarkerSize', 8,...
302     'tag', 'Dynamics');
303
304 LogID = (DATA.Clusters.Statistics(:,2) < DATA.ConfidenceInterval.Lower);
305 handles.Plots.Unknown = plot(...
306     DATA.Clusters.Centers(LogID, 1),...
307     DATA.Clusters.Statistics(LogID, 2),...
308     'k^',...
309     'LineWidth', 1.5,...
310     'MarkerFaceColor', 'g',...
311     'MarkerSize', 8,...
312     'tag', 'Unknown');
313
314
315
316 hold(handles.Subplot.BVA.Axes, 'off');
317
318 % ----- %
319 % [6] Compute Histograms:
320
321 DATA.Histogram.Edges{1,1} = -0.025 : 0.05 : 1.025; % Comment out to retain ...
322     3D hist edges
323
324 Counts1 = histc(DATA.Results(:,1),...
325     DATA.Histogram.Edges{1,1});
326
327 handles.Subplot.Histogram.Mean.Plots.Bar =...
328     bar(handles.Subplot.Histogram.Mean.Axes,...
329     DATA.Histogram.Edges{1,1}, Counts1, 'histc');
330
331 Counts2 = histc(DATA.Results(:,2),...
332     DATA.Histogram.Edges{1,2});

```

```

333 handles.Subplot.Histogram.StDev.Plots.Bar =...
334     barh(handles.Subplot.Histogram.StDev.Axes,...
335         DATA.Histogram.Edges{1,2}, Counts2, 'histc');
336
337 % ----- %
338 %% Adjust Axes:
339
340 % BVA:
341 xlim(handles.Subplot.BVA.Axes,...
342     [min(Config.ShotNoise.Prediction.OrdinateVals),...
343     max(Config.ShotNoise.Prediction.OrdinateVals)]);
344
345 ylim(handles.Subplot.BVA.Axes,...
346     [0, max(DATA.Results(:,2))]);
347
348 set(handles.figure,...
349     'color', Config.Figure.BackgroundColor)
350
351 ylabel(handles.Subplot.BVA.Axes,...
352     Config.Axes.YLabel.String,...
353     'FontSize', Config.Axes.YLabel.FontSize);
354
355 xlabel(handles.Subplot.BVA.Axes,...
356     Config.Axes.XLabel.String,...
357     'FontSize', Config.Axes.XLabel.FontSize)
358
359 title(handles.Subplot.BVA.Axes,...
360     Config.Axes.Title.String,...
361     'FontSize', Config.Axes.Title.FontSize);
362
363 set(handles.Subplot.BVA.Axes,...
364     'TickDir', Config.Axes.TickDir,...
365     'Box', Config.Axes.Box);
366
367
368 % Histograms:
369 xlim(handles.Subplot.Histogram.Mean.Axes,...
370     [min(Config.ShotNoise.Prediction.OrdinateVals),...
371     max(Config.ShotNoise.Prediction.OrdinateVals)]);
372
373 ylabel(handles.Subplot.Histogram.Mean.Axes,...
374     Config.Histogram.Mean.Axes.YLabel.String,...
375     'FontSize', Config.Histogram.Mean.Axes.YLabel.FontSize);
376
377 set(handles.Subplot.Histogram.Mean.Axes,...
378     'Box', Config.Histogram.Mean.Axes.Box,...
379     'TickDir', Config.Histogram.Mean.Axes.TickDir);
380
381

```

```

382 ylim(handles.Subplot.Histogram.StDev.Axes,...
383       [0, max(DATA.Results(:,2))]);
384
385 xlabel(handles.Subplot.Histogram.StDev.Axes,...
386        Config.Histogram.StDev.Axes.XLabel.String,...
387        'FontSize', Config.Histogram.StDev.Axes.XLabel.FontSize);
388
389 set(handles.Subplot.Histogram.StDev.Axes,...
390     'Box', Config.Histogram.StDev.Axes.Box,...
391     'TickDir', Config.Histogram.StDev.Axes.TickDir);
392
393
394 % ----- %
395 % Create a colorbar:
396
397 APOS = get(handles.Subplot.BVA.Axes, 'position');
398
399 handles.Colorbar = colorbar('peer', handles.Subplot.BVA.Axes);
400
401 CPOS = get(handles.Colorbar, 'position');
402
403 set(handles.Colorbar,...
404     'Position', [APOS(1)+APOS(3)+0.01, CPOS(2), CPOS(3), CPOS(4)]);
405
406 % ----- %
407 % Configure the Legend:
408 % handles.Plots.Current = [handles.Plots.Contour, ...
409 %     handles.Plots.ShotNoisePrediction,...
410 %     handles.Plots.ConfidenceInterval.Upper, ...
411 %     handles.Plots.ConfidenceInterval.Lower,...
412 %     handles.Plots.ConfidenceInterval.Mean,...
413 %     handles.Plots.Static, handles.Plots.Dynamics, handles.Plots.Unknown];
414
415 % handles.Legend = legend(...
416 %     handles.Plots.Current,...
417 %     get(handles.Plots.Current, 'tag'));
418
419 % set(handles.Legend,...
420 %     'Box', DATA.BVA.Config.Legend.Box,...
421 %     'Location', DATA.BVA.Config.Legend.Location);
422
423 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
424 %% EOF
425
426
427
428 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
429 %% StartUp:
430 function StartUp()
431 %
432 %
433 %
434
435 % Add path(s):
436 addpath(...
437     'Other/histcn',...
438     'Other/smoothn');
439
440
441
442
443
444 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
445 %% CleanUp:
446 function CleanUp()
447 %
448 %
449 %
450
451 % Remove path(s):
452 rmpath(...
453     'Other/histcn',...
454     'Other/smoothn');
```