Burst Variance Analysis: Help & Documentation

Kristofer Gryte[1]

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

October 21, 2011

Contents

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:
  - `InitalizeAxes.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 Nx3 or 3xN, 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:

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

- **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-α) 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 ALEx; for single-laser excitation, ALEx 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 $\left\{\mu^{E^*} : a \leq \mu^{E^*} < b\right\}$, 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: Nx3; N = total photons
Input: *WINDOW*
Output: *DATA*

Var: *C = CLUSTERBINS;* α *= CI; W = 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 $E_j^* = F_j^A/L$; $F^A = \Sigma_{i=1}^{W}(w_i == 2)$
end
∘ Calculate $\mu_k^{E^*}$;
∘ Calculate $\sigma_k^{E^*}$;
∘ Determine $\mu_k^{E^*} \in \forall C_i$;
end

for $i := 1$ to C

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

DATA ← RESULTS;

Visualize *DATA*;

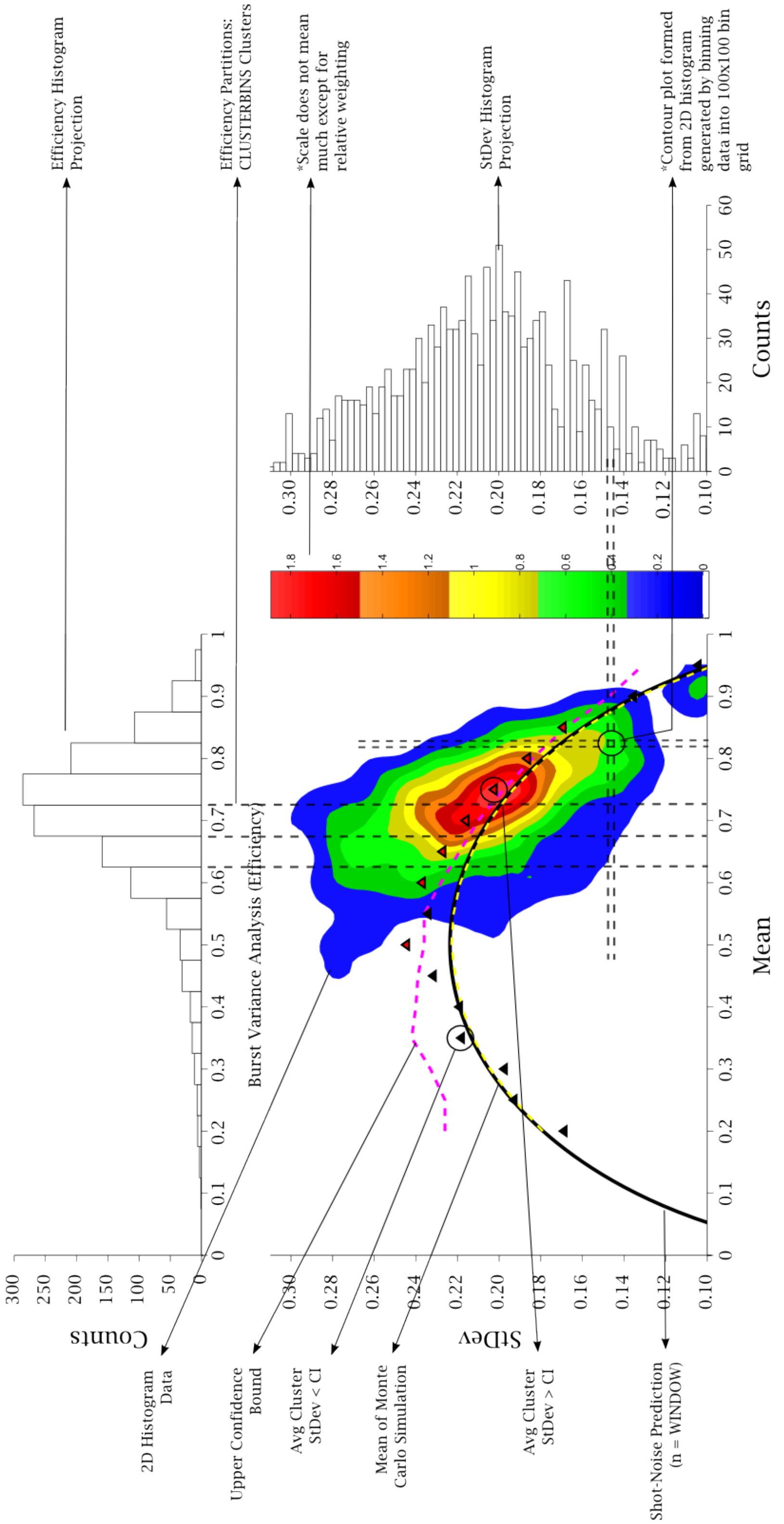
Visualization

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.



Efficiency Histogram Projection
Efficiency Partitions: CLUSTERBINS Clusters
*Scale does not mean much except for relative weighting
StDev Histogram Projection
*Contour plot formed from 2D histogram generated by binning data into 100x100 bin grid
Burst Variance Analysis (Efficiency)
Counts
Mean
StDev
2D Histogram Data
Upper Confidence Bound
Avg Cluster StDev < CI
Mean of Monte Carlo Simulation
Avg Cluster StDev > CI
Shot-Noise Prediction (n = WINDOW)

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:

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

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

where n = *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 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

Code

BVA Script

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

```
%                 0.52, 1,    1;...
%                 0.53, 0,    1;...
%                 0.54, 2,    1;...
%                 0.55, 0,    1;...
%                 0.62, 3, NaN;...
%                    .   .     .
%                    .   .     .
%                    .   .     .
%                51.70, 2, NaN;...
%                51.74, 0, 100;...
%                51.75, 2, 100;...
%                51.76, 2, 100;...
%                51.77, 1, 100;...
%                51.78, 1, 100;...
%                52.81, 2, NaN];
%
%   In the 'Testing' section, an example demonstrates how an Arrival Matrix
%   containing only arrival times and photon IDs can be transformed to
%   accommodate burst search input, as well as highlights some
%   considerations regarding the data (e.g., long bursts being split into
%   two, resulting in two adjacent bursts sharing the same photon, and the
%   need to increasing end times by a fractional amount to ensure
%   appropriate binning [NOTE: this, of course, can be avoided using
%   a binning method other than HISTC.M]).
%
%   Following the 'Testing' section, find the 'Initial Parameters' section,
%   which provides a starting point for input parameters. The BVA parameters
%   are typical values, as detailed in Torella, et al (2011); whereas, the
%   filter parameters will vary depending on the experimental setup and
%   conditions. For the example data included, the filter values are
%   found appropriate.
%
%   And finally, in the 'BVA' section, the figure generation, analysis, and
%   plotting is broken down into 4 steps (functions), with example usage
%   included.
%
%   For additional questions and concerns regarding the individual
%   functions themselves, please consult each function's HELP. Importantly,
%   more detail concerning the nature and structure of the input and output
%   parameters is provided. Further information concerning BVA
%   interpretation may be found in the documentation provided: HELP.pdf .
%
%   Lastly, if any bugs or breaks are found in the code, please contact the
%   author of this package detailing the problem and provide any additional
%   evidence of the error/problem, which includes, but is not limited to,
%   error messages, figures, improper output data structure, et cetera. The
%   author's contact information may be found below.
%
%   NOTES:
```

```
%       [1] Need STATS toolbox for generating random numbers drawn from a
%       binomial distribution.
%
%   ----------------------
%   Publications:
%       [1] Torella, JP, Holden, SJ, Santoso, Y, Hohlbein, J, Kapanidis, AN
%       (2011). Identifying molecular dyanmics in single-molecule FRET
%       experiments with burst variance analysis. Biophys J. 100(6):1568-77.
%
%   NOTICE:
%       If this software is used in the analysis of data which is to be
%       presented in publication, please cite the above work and
%       acknowledge the authors of this software.
%
%       For purposes of redistribution, use, and/or promotion of products
%       derived, consult the included license agreement.
%
%       Additionally, further license agreements are contained in
%       specialized subfunctions included in the directory 'Other'. Use,
%       redistribution, and derivative works are subject to any additional
%       requirements imposed therein.
%
%   Copyright (c) 2011. Kristofer Gryte. University of Oxford.
%
%
%
%   Version: 1.0 (2011-08-04)
%
%   ----------------------
%   Author Information:
%       Kristofer Gryte
%       Clarendon Laboratory
%       University of Oxford
%       Oxford, United Kingdom
%       e-mail: k.gryte1 (at) physics.ox.ac.uk
%
%   ----------------------
%   TODO:
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOP


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Testing:
```

```

addpath(...
    'Other/lightspeed',...
    'Other/SplitVec');

clear ArrivalMatrix StartTimes EndTimes

load('MyArrivalMatrix.mat');

% Check!!!! Before binning photons, need to ensure that no StartTimes
% and EndTimes are equal (i.e., no burst ends when another begins), as
% this creates problems when creating the edges (eliminates a non-burst
% bin).
AdjacentBursts = intersect_sorted(...
    StartTimes,...
    EndTimes);

if isempty(AdjacentBursts) == false
    % Adjacent bursts have been found!!!
    fprintf('%d bursts were found to be immediately adjacent.\n',...
        numel(AdjacentBursts));

    % Cycle through each Adjacent Burst:
    for ADJBURST = 1 : numel(AdjacentBursts)

        % Find the burst:
        AdjBurstIndex = find(...
            EndTimes == AdjacentBursts(ADJBURST),...
            1,...
            'first');

        % Increase the arrival time of any photons which
        % arrived at the previous start time by a fractional amount:
        ArrivalMatrix(ArrivalMatrix(:,1) == EndTimes(AdjBurstIndex, 1), 1) ...
            =...
            ArrivalMatrix(ArrivalMatrix(:,1) == EndTimes(AdjBurstIndex, ...
                1), 1)...
            +...
            1e-10; % should be less than NI card time resolution...

        % Increase the start time of the next burst by a fractional amount:
        EndTimes(AdjBurstIndex, 1) =...
            EndTimes(AdjBurstIndex, 1)...
            +...
            1e-10;

    end % end FOR

```

```
end % end IF


% Configure the ArrivalMatrix:
Edges = union_sorted_rows(...
    StartTimes,...
    EndTimes + 1e-10); % Bump the end times by a fractional amount so that ...
       last photon can be placed in appropriate burst bin.

% Histogram the arrival times:
[Counts, BinIndices] = histc(ArrivalMatrix(:,1), Edges);

% Determine which bins to throw away: (e.g., all those bins containing
% photons which do not belong to a burst)

% Discard the even BinIndices, as these correspond to bins defined by
% EdgeA = EndTime(1); EdgeB = StartTime(2). Hence, this bin (EdgeA,
% EdgeB) corresponds to photons which do not belong to a burst (i.e., photons
% occurring after the end of the previous burst and before the start of
% the next burst):
VALS1 = 0:2:max(BinIndices); % Note: we include 0 to account for photons ...
   not falling into any defined bin.
TF = ismember_sorted(BinIndices, VALS1);


% Remove all photons not belonging to a burst, thus creating a 'burst
% arrival matrix':
BinIndices(TF == true) = NaN; %[];

% Tack on the BinIndices (i.e., Burst Indices) to the Arrival Matrix:
ArrivalMatrix(:,3) = BinIndices;

clear BinIndices TF VALS1 Counts Edges


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initial Parameters:

WINDOW = 5;
CLUSTERBINS = 20;
CLUSTERMINWINDOWS = 50;
CONFIDENCEINTERVAL = 0.999;

% ---------- %
%  Filters:
% ---------- %

% DexDem:
FILTERS.DexDem.Min = [];
```

```
FILTERS.DexDem.Max = [];

% DexAem:
FILTERS.DexAem.Min = [];
FILTERS.DexAem.Max = [];

% AexAem:
FILTERS.AexAem.Min = [30];
FILTERS.AexAem.Max = [];

% Sum Photons: DexDem + DexAem
FILTERS.SumPhotons.Min = [];
FILTERS.SumPhotons.Max = [];

% Total Photons: DexDem + DexAem + AexAem
FILTERS.TotalPhotons.Min = [];
FILTERS.TotalPhotons.Max = [];

% Efficiency:
FILTERS.Efficiency.Min = [];
FILTERS.Efficiency.Max = [];

% Stoichiometry:
FILTERS.Stoichiometry.Min = [0.45];
FILTERS.Stoichiometry.Max = [];

% Duration: [seconds]
FILTERS.Duration.Min = [];
FILTERS.Duration.Max = [];

% Length: [photons]
FILTERS.Length.Min = [];
FILTERS.Length.Max = [];



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% BVA:

% [1] Create a figure:
handles.figure = figure;

set(handles.figure,...
    '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.

```

```
% [2] Initialize the figure and axes:
handles = InitializeAxes(handles);

% [3] Perform BVA:
[DATA] = Confocal2ColorALExSolnMultDataBVA_Analysis(...
    ArrivalMatrix,...
    'window', WINDOW,...
    'clusterbins', CLUSTERBINS,...
    'clusterminwindows', CLUSTERMINWINDOWS,...
    'confidenceinterval', CONFIDENCEINTERVAL,...
    'filters', FILTERS);

% [4] Plot the results:
handles = PlotResults(DATA, WINDOW, handles);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOS
```

Initialize Axes

```
function handles = InitializeAxes(handles)
% INITIALIZEAXES
%   Initialize axes within a figure window according to defined
%   specifications. Three subplots are generated: [i] 2D histogram subplot
%   and [ii] 2 1D histogram suplots along the upper and right edges of the
%   2D histogram subplot. Orientations of the 1D histogram subplots are
%   such that they are the projections of the data along the abscissa and
%   ordinate axes, respectively.
%
%   INPUTS:
%       HANDLES: structure with the following fields:
%          FIGURE: handle to the figure window. Suggested figure size:
%          [520, 196, 900, 700].
%
%
%   OUTPUTS:
%       HANDLES: structure with the following fields:
%           SUBPLOT: nested structure with the following fields:
%               BVA: nested structure with the following fields:
%                   AXES: handles to the 2D Histogram axes (subplot).
%           SUBPLOT: nested structure with the following fields:
%               HISTOGRAM: nested structure with the following fields:
%                   MEAN: nested structure with the following fields:
%                      AXES: handle to the axes into which the histogram
%                         will be plotted.
%                   STDEV: nested structure with the following fields:
%                      AXES: handle to the axes into which the histogram
%                         will be plotted.
%
%
%
%   DEPENDENCIES:
%       (none)
%
%   -----------------------
%   EXAMPLE USAGE:
%       handles = InitializeAxes(handles);
%
%   ------------------------
%   History:
%       [1] Kristofer Gryte. University of Oxford (2011).
%
%   Copyright (c) 2011. Kristofer Gryte. University of Oxford.
%
%
```

```
%
%   Version: 1.0 (2011-08-02)
%
%   ----------------------
%   Author Information:
%       Kristofer Gryte
%       Clarendon Laboratory
%       University of Oxford
%       Oxford, United Kingdom
%       e-mail: k.gryte1 (at) physics.ox.ac.uk
%
%   ------------------------
%   TODO:
%
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOP


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialization:

% Activate figure window:
figure(handles.figure);

% Configure the figure colormap:
load('BVAcolormap.mat', 'BVAcolormap');

set(handles.figure,...
    'Colormap', BVAcolormap);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialize Subplots:

% Mean Histogram Subplot:
handles.Subplot.Histogram.Mean.Axes = subplot(3,1,1);

% Standard Deviation Histogram Subplot:
handles.Subplot.Histogram.StDev.Axes = subplot(3,1,2);

% BVA Subplot:
handles.Subplot.BVA.Axes = subplot(3,1,3);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Manipulate Subplots:
```

```

% Get the current units:
OldUnits = get(handles.Subplot.Histogram.Mean.Axes, 'Units');

% Set Units: % Note: Change to characters...as the number of pixels is
% relative to user monitor...thus, standardize...
set([handles.Subplot.Histogram.Mean.Axes,...
    handles.Subplot.Histogram.StDev.Axes,...
    handles.Subplot.BVA.Axes],...
    'Units', 'pixels');

% Set Mean Histogram Position:
set(handles.Subplot.Histogram.Mean.Axes,...
    'Position', [150 500 350 150]);

% Set StDev Histogram Position:
set(handles.Subplot.Histogram.StDev.Axes,...
    'Position', [600 95 150 350]);

% Set BVA Subplot Position:
set(handles.Subplot.BVA.Axes,...
    'Position', [150 95 350 350]);

% Return units to 'characters':
set([handles.Subplot.Histogram.Mean.Axes,...
    handles.Subplot.Histogram.StDev.Axes,...
    handles.Subplot.BVA.Axes],...
    'Units', OldUnits); % Maybe Normalize?


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOF
```

BVA Analysis

```
function [DATA] = Confocal2ColorALExSolnMultDataBVA_Analysis(...
    varargin)
% CONFOCAL2COLORALEXSOLNMULTDATABVA_ANALYSIS
%   Burst Variance Analysis (BVA) of confocal (2 color) solution
%   alternating laser excitation (ALEx) data.
%
%   INPUTS:
%       ARRIVALMATRIX: input data on which to perform Burst Variance
%       Analysis. The input matrix should be Nx3 or 3xN, 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 a
%       corresponding index to a unique burst ID; background photons should
%       be indexed as NaN): ARRIVALMATRIX = ArrivalTimes   ID   BurstNumber
%
%       WINDOW: size of the sliding window 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 deviation
%       willl be calculated across bursts belonging to a particular bin and
%       compared 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 to be analyzed
%       (i.e., we need to make sure we have enough data to be worth our
%       while). Default: 50.
%
%       CONFIDENCEINTERVAL: (1-alpha) to determine statistical signifance
%       of cluster results. Default: 0.999.
%
%       FILTERS: nested 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.
%                (excluding photons detected in donor channel upon acceptor
%                excitation: AexDem)
%            EFFICIENCY: the FRET efficiency.
%            STOICHIOMETRY: the relative fluorophore brightness.
%
%    OUTPUTS:
%        DATA: structure with the following data streams for each detected
%        burst: (NOTE: data streams are filtered according to input FILTERS)
%            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.
%                (excluding photons detected in donor channel upon acceptor
%                excitation: AexDem)
%            EFFICIENCY: the FRET efficiency.
%            STOICHIOMETRY: the relative fluorophore brightness.
%            RESULTS: an Nx2 array, where N is the number of bursts (after
%                filtering). The first column corresponds to the mean ...
Efficiency
%                value for a burst and the second to the burst's Efficiency
%                standard deviation. Array: E(efficiency)   sqrt(V(efficiency))
%            CLUSTERS: nested structure with the following fields:
%                CENTERS: vector containing the centers of the individual
%                   clusters.
%                STDEVDISTRMEAN: the Monte Carlo simulated standard
%                   deviation prediction.
%                STATISTICS: an Mx2 array, where M is the number of clusters
%                    ('slices' along the efficiency axis, into which bursts are
%                    grouped). The first column corresponds to the mean
%                    Efficiency value for the cluster, and the second to the
%                    standard deviation of the burst's within that cluster.
%                WINDOWMEANS: cellular array where each entry contains the
%                    the Efficiency values for all sliding windows for bursts
%                    belonging to a cluster.
%                DISTANCEFROMUPPERCI: a vector of the distance between the
%                    mean standard deviations of each cluster from the upper
%                    confidence bound. This expresses something about the ...
extent
%                    of the 'dynamics'.
%            CONFIDENCEINTERVAL: nested structure with the following fields:
```

```
%               UPPER: the upper confidence bound generated from the Monte
%                  Carlo simulation and multiple hypothesis test.
%               LOWER: the lower confidence bound generated from the Monte
%                  Carlo simulation and the multiple hypothesis test.
%
%   DEPENDENCIES:
%       SPLITVEC.M
%       INTERSECT_SORTED.M (Lightspeed)
%       UNION_SORTED_ROWS.M (Lightspeed)
%       ISMEMBER_SORTED.M (Lightspeed)
%       SETDIFF_SORTED.M (Lightspeed)
%       FILTERDATA.M (private)
%       MONTECARLOPREDICTION.M (private; requires STATS toolbox)
%
%   -----------------------
%   EXAMPLE USAGE:
%       Confocal2ColorALExSolnMultDataBVA_Analysis(ArrivalMatrix);
%
%       Confocal2ColorALExSolnMultDataBVA_Analysis(ArrivalMatrix, 'window',
%       5);
%
%       Confocal2ColorALExSolnMultDataBVA_Analysis(ArrivalMatrix, 'window',
%       5, 'clusterbins', 20, 'clusterminwindows', 50);
%
%   ------------------------
%   History:
%       [1] Joseph Torella, Yusdi Santoso. University of Oxford (2009).
%       [2] Johannes Hohlbein. University of Oxford (2010).
%       [3] Kristofer Gryte. University of Oxford (2011).
%
%   Publications:
%       [1] Torella, JP, Holden, SJ, Santoso, Y, Hohlbein, J, Kapanidis, AN
%       (2011). Identifying molecular dyanmics in single-molecule FRET
%       experiments with burst variance analysis. Biophys J. 100(6):1568-77.
%
%   NOTICE:
%       If this software is used in the analysis of data which is to be
%       presented in publication, please cite the above work and
%       acknowledge the authors of this software.
%
%   Copyright (c) 2011. Kristofer Gryte. University of Oxford.
%
%
%
%   Version: 1.0 (2011-07-29)
%
%   ----------------------
%   Author Information:
%       Kristofer Gryte
```

```
%       Clarendon Laboratory
%       University of Oxford
%       Oxford, United Kingdom
%       e-mail: k.grytel (at) physics.ox.ac.uk
%
%   ------------------------
%   TODO:
%       [1] Provide appropriate checks on filter values (e.g., ensure
%       numeric, positive, etc.)
%       [2] Make number of Monte Carlo simulations an input variable?
%
%   ------------------------
%   See also
%
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOP



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% StartUp/CleanUp:

StartUp();

C = onCleanup(@() CleanUp());


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialization:

% ---- %
% Establish default values:
% ---- %
WINDOW = 5; % size of the sliding window (non-overlapping) over which FRET ...
    will be calculated; hence, for every 5 photons, we calculate FRET
CLUSTERBINS = 20; % number of bins over which we will cluster bursts ...
    characterized by their mean FRET value
CLUSTERMINWINDOWS = 50; % threshold for the minimum number of total ...
    windows within a cluster over all bursts for the cluster to be ...
    analyzed (i.e., we need to make sure we have enough data to be worth ...
    our while)

CONFIDENCEINTERVAL = 0.999; % (1-alpha) CI level to determine if observed ...
    variance is statistically significant
NUMMONTECARLOSAMPLES = 500; % Number of Monte Carlo simulations to run for ...
    CI bound generation

```

```
FILTERS = struct(...
    'DexDem',        struct('Min', [], 'Max', []),...
    'DexAem',        struct('Min', [], 'Max', []),...
    'AexAem',        struct('Min', [], 'Max', []),...
    'AexDem',        struct('Min', [], 'Max', []),...
    'Efficiency',    struct('Min', [], 'Max', []),...
    'Stoichiometry',struct('Min', [], 'Max', []),...
    'SumPhotons',    struct('Min', [], 'Max', []),...
    'TotalPhotons', struct('Min', [], 'Max', []),...
    'Duration',      struct('Min', [], 'Max', []),...
    'Length',        struct('Min', [], 'Max', []));

DATA = [];

% ---- %
% Parse input arguments:
% ---- %

ArrivalMatrix = varargin{1};

% Check!!!
if ¬isnumeric(ArrivalMatrix)
    % error out!
    errordlg(['ERROR:invalid input argument for BVA function. Please input ...
       ',...
        'a numeric array for the initial argument.'],...
        'ERROR:invalid input argument');
    return;
end % end IF

% Perform check for data input size and ensure dimensionality is correct
% (column matrix)
if size(ArrivalMatrix,1) < size(ArrivalMatrix, 2) % More columns than rows
    % Take the tranpose:
    ArrivalMatrix = ArrivalMatrix';

elseif size(ArrivalMatrix,2) ≠ 3 % too many or too few columns

    % Error out!
    errordlg(['ERROR:invalid input argument for BVA function. Data input ',...
        'should be either (Nx3) or (3xN).'],...
        'ERROR:invalid input argument');
    return;

end % end IF/ELSEIF

% Check number of input arguments:
if (mod(nargin, 2) - 1) ≠ 0 % odd number of input arguments
    % Throw a fit and error out!
```

```
    errordlg(['ERROR:invalid number of input arguments for BVA function. ',...
        'Please enter parameter-value pairs in addition to the data ...
            input.'],...
        'ERROR:invalid number of input arguments');
    return;

end % end IF

% Loop through and assign parameter-value pairs:
for NumArg = 2 : 2 : nargin

    switch lower(char(varargin{NumArg}))

        case {'win', 'window', 'winsize'}
            % Update the WINDOW size:
            temp = varargin{NumArg + 1};

            % Check!
            if ¬isnumeric(temp)   temp ≤ 1
                % Error out:
                answer = questdlg(...
                    ['NOTICE:An invalid value was provided for the WINDOW ...
                        paramter. ',...
                    'Please supply an integer value greater than 1. ',...
                    'Would you like to continue with a default WINDOW = 5, ...
                        ',...
                    'or return to the command-line and supply an ...
                        appropriate input value?'],...
                    'WARNING:invalid input value',...
                    'Continue', 'Cancel',...
                    'Cancel');

                % Parse the answer:
                switch lower(answer)
                    case 'continue'
                        % Move along with default WINDOW value
                    case 'cancel'
                        % Abort!
                        return;
                end % end SWITCH

            else
                % Assign over the input WINDOW size:
                WINDOW = temp;
            end % end IF/ELSE

        case {'clusterbins', 'bins', 'numbins', 'binnumber', 'binnum'}

```

```
        % Update the number of cluster bins:
        temp = varargin{NumArg + 1};

        % Check!
        if ¬isnumeric(temp)   temp ≤ 1
            % Error out:
            answer = questdlg(...
                ['An invalid value was provided for the CLUSTERBINS ...
                   paramter. ',...
                'Please supply an integer value greater than 1. ',...
                'Would you like to continue with a default CLUSTERBINS ...
                   = 20, ',...
                'or return to the command-line and supply an ...
                   appropriate input value?'],...
                'WARNING:invalid input value',...
                'Continue', 'Cancel',...
                'Cancel');

            % Parse the answer:
            switch lower(answer)
                case 'continue'
                    % Move along with default CLUSTERBINS value
                case 'cancel'
                    % Abort!
                    return;
            end % end SWITCH

        else
            % Assign over the input CLUSTERBINS value:
            CLUSTERBINS = temp;
        end % end IF/ELSE

    case {'clusterminwindows', 'minwindows'}

        % Update the minimum number of windows within a cluster:
        temp = varargin{NumArg + 1};

        % Check!
        if ¬isnumeric(temp)   temp ≤ 1
            % Error out:
            answer = questdlg(...
                ['An invalid value was provided for the ...
                   CLUSTERMINWNDOWS paramter. ',...
                'Please supply an integer value greater than 1. ',...
                'Would you like to continue with a default ...
                   CLUSTERMINWINDOWS = 50, ',...
                'or return to the command-line and supply an ...
                   appropriate input value?'],...
                'WARNING:invalid input value',...
```

```
                    'Continue', 'Cancel',...
                    'Cancel');

                % Parse the answer:
                switch lower(answer)
                    case 'continue'
                        % Move along with default CLUSTERMINWINDOWS value
                    case 'cancel'
                        % Abort!
                        return;
                end % end SWITCH

            else
                % Assign over the input CLUSTERMINWINDOWS:
                CLUSTERMINWINDOWS = temp;
            end % end IF/ELSE

        case {'ci', 'confidenceinterval', 'conf', 'confidence', ...
            'confinterval'}

            % Update the confidence interval:
            temp = varargin{NumArg + 1};

            % Check!
            if ¬isnumeric(temp)    (temp ≥ 1    temp ≤ 0)
                % Error out:
                answer = questdlg(...
                    ['An invalid value was provided for the ...
                        CONFIDENCEINTERVAL paramter. ',...
                    'Please supply a numeric value greater than 0 and less ...
                        than 1. ',...
                    'Would you like to continue with a default ...
                        CONFIDENCEINTERVAL = 0.999, ',...
                    'or return to the command-line and supply an ...
                        appropriate input value?'],...
                    'WARNING:invalid input value',...
                    'Continue', 'Cancel',...
                    'Cancel');

                % Parse the answer:
                switch lower(answer)
                    case 'continue'
                        % Move along with default CONFIDENCEINTERVAL value
                    case 'cancel'
                        % Abort!
                        return;
                end % end SWITCH

            else
```

```
                % Assign over the input CONFIDENCEINTERVAL:
                CONFIDENCEINTERVAL = temp;
            end % end IF/ELSE

        case {'filters', 'filter', 'thresh', 'thresholds', 'threshold', ...
            'filt'}

            % Update the FILTERS:
            temp = varargin{NumArg + 1};

            % Check!
            if ¬isstruct(temp)
                % Error out:
                answer = questdlg(...
                    ['An invalid value was provided for the FILTERS ...
                        paramter. ',...
                    'Please supply an appropriate structure containing ...
                        relevant fields. ',...
                    'Would you like to continue with a default FILTERS ...
                        (none), ',...
                    'or return to the command-line and supply an ...
                        appropriate input?'],...
                    'WARNING:invalid input value',...
                    'Continue', 'Cancel',...
                    'Cancel');

                % Parse the answer:
                switch lower(answer)
                    case 'continue'
                        % Move along with default FILTERS value
                    case 'cancel'
                        % Abort!
                        return;
                end % end SWITCH

            else
                % Assign over the input FILTERS:
                FILTERS = temp;
            end % end IF/ELSE

        otherwise
            % Throw a notice that the input argument was not recognized:
            msgbox(sprintf(...
                ['NOTICE:Unrecognized input argument: %s. \n',...
                    'Parameter-value pair ignored']),...
                char(varargin{NumArg}));

    end % end SWITCH

```

```
end % end FOR


% ---- %
% Calculate remaining parameters:
PHOTONCUTOFF = floor(WINDOW/2); % Remove the first few photons of a burst ...
    (allows analyzed data to be centered within burst)

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.

% ---- %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Filtering:

% Filter the data:
[DATA, REMOVEBURSTS] = FilterData(ArrivalMatrix, FILTERS); % this is a ...
    function!!!!



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Burst Variance Analysis:

% Start the timer...
tic

% Launch the waitbar:
h.Waitbar = waitbar(...
    0,...
    'Commencing BVA...please be patient');

% ----------------------------------------- %
%% Winnow the Data:

% Replace all the NaNs:
ArrivalMatrix(isnan(ArrivalMatrix(:,3)),3) = 0;

% Return the assigned IDs of the various runs: % recall that non-burst
% photons are labeled with a '0'!!!!
FLAGVALS = SplitVec(...
    ArrivalMatrix(:,3),...ArrivalMatrix(¬isnan(ArrivalMatrix(:,3)),3),... ...
        % exclude any NaNs
    'equal',... % find all those runs of the same index (burst ID)
    'firstval'); % return just the first index marking a run.

% Extract some information about the bursts from the ArrivalMatrix:
```

```
STARTINDICES = SplitVec(...
    ArrivalMatrix(:,3),... % exclude any NaNs
    'equal',... % find all those runs of the same index (burst ID)
    'first'); % return just the first index marking a run.

ENDINDICES = SplitVec(...
    ArrivalMatrix(:,3),... % exclude any NaNs
    'equal',... % find all those runs of the same index (burst ID)
    'last'); % return just the last index completing a run.

STARTINDICES(FLAGVALS == 0) = []; % NOTE:
ENDINDICES(FLAGVALS == 0) = [];

% Get the start times and end times for each burst:
STARTTIMES = ArrivalMatrix(STARTINDICES, 1);
ENDTIMES = ArrivalMatrix(ENDINDICES, 1);

% Grab the total number of bursts:
TOTALBURSTS = numel(STARTINDICES);

% Create an index array:
BURSTINDICES(:,1) = 1 : TOTALBURSTS; % column vector

% Check!!!! Before binning photons, need to ensure that no StartTimes
% and EndTimes are equal (i.e., no burst ends when another begins), as
% this creates problems when creating the edges (eliminates a non-burst
% bin).
AdjacentBursts = intersect_sorted(...
    STARTTIMES,...
    ENDTIMES);

if isempty(AdjacentBursts) == false
    % Adjacent bursts have been found!!!
    fprintf('%d bursts were found to be immediately adjacent.\n',...
        numel(AdjacentBursts));

    % Cycle through each Adjacent Burst:
    for ADJBURST = 1 : numel(AdjacentBursts)

        % Find the burst:
        AdjBurstIndex = find(...
            STARTTIMES == AdjacentBursts(ADJBURST),...
            1,...
            'first');

        % Increase the arrival time of any photons which
        % arrived at the previous start time by a fractional amount:
        ArrivalMatrix(ArrivalMatrix(:,1) == STARTTIMES(AdjBurstIndex, 1), ...
           1) =...
```

```
            ArrivalMatrix(ArrivalMatrix(:,1) == STARTTIMES(AdjBurstIndex, ...
                1), 1)...
            +...
            1e-10; % should be less than NI card time resolution...

        % Increase the start time of the next burst by a fractional amount:
        STARTTIMES(AdjBurstIndex, 1) =...
            STARTTIMES(AdjBurstIndex, 1)...
            +...
            1e-10;

    end % end FOR


end % end IF


% Create Edges:
Edges = union_sorted_rows(...
    STARTTIMES(:,1),...
    ENDTIMES(:,1) + 1e-10);

% Histogram the arrival times:
[Counts, BinIndices] = histc(ArrivalMatrix(:,1), Edges);

% Determine which bins to throw away: (e.g., all those bins containing
% photons which do not belong to a burst)

% Discard the even BinIndices, as these correspond to bins defined by
% EdgeA = EndTime(1); EdgeB = StartTime(2). Hence, this bin (EdgeA,
% EdgeB) corresponds to photons which do not belong to a burst (i.e., photons
% occurring after the end of the previous burst and before the start of
% the next burst):
VALS1 = 0:2:max(BinIndices); % Note: we include 0 to account for photons ...
    not falling into any defined bin.
TF = ismember_sorted(BinIndices, VALS1);


% Remove all photons not belonging to a burst, thus creating a 'burst
% arrival matrix':
ArrivalMatrix(TF == true,:) = [];
BinIndices(TF == true) = [];


% ---------------------------------------- %
%% Burst Removal:

% Remove the filtered bursts:
TF = ismember_sorted((BinIndices + 1) ./ 2, REMOVEBURSTS);
```

```

ArrivalMatrix(TF == true, :) = [];
BinIndices(TF == true, :) = [];


% Now, we need to rid ourselves of all photons originating from red
% excitation: DexDem = 0; DexAem = 2; AexAem = 1; AexDem = 3; => all
% photons labeled 1 or 3... (NOTE: this also removes acceptor-only
% bursts!)
LogicalIndexArray = (ArrivalMatrix(:,2) == 1   ArrivalMatrix(:,2) == 3);
VALS2 = unique(BinIndices(LogicalIndexArray, :)); % This gives all burst ...
    indices which contain acceptor excitation photons

ArrivalMatrix(LogicalIndexArray,:) = []; % All acceptor excitation photons ...
    removed.
BinIndices(LogicalIndexArray,:) = []; % Note: this array will be important ...
    now, as all photons can be grouped by their unique identification with ...
    a Bin...
% NOTE: now VALS and unique(BinIndices) do not necessarily
% correspond!!!

% Determine which, if any, bursts have been removed from further
% analysis:
REMOVEBURSTS = [REMOVEBURSTS; (setdiff_sorted(VALS2, unique(BinIndices)) + ...
    1) ./ 2];

% REMOVEBURSTS now contains all bursts which either did not meet the
% filtering criteria above or were acceptor-only...

% Frequently, bursts have values which are not numbers; find them:
NaNBURSTS = find(isnan(DATA.Efficiency(:,1)));

REMOVEBURSTS = [REMOVEBURSTS; NaNBURSTS];





% ---------------------------------------- %
% Clustering: cluster the bursts according to mean Efficiency values:

% Define the cluster bounds: % NOTE:this creates an extra bin!!!
CLUSTEREDGES = -CLUSTERWIDTH/2:CLUSTERWIDTH:1+CLUSTERWIDTH/2;
CLUSTEREDGES = CLUSTEREDGES';

% Each burst is histogrammed into a cluster bin:
[Counts, CLUSTERBINIDX] = histc(...
    DATA.Efficiency(:,1),...
    CLUSTEREDGES);
```

```
% ---------------------------------------- %

% Split the Arrival Matrix into Bursts:
IndexCell = SplitVec(...
    BinIndices,...
    'equal',... % Group all elements which are equal
    'loc'); % Return the locations of these elements in a cellular array

% Each cellular element of IndexCell corresponds to a burst; within each
% cell element, the individual photons are assigned a location
% corresponding to their index in the (filtered) ArrivalMatrix.


% ---------------------------------------- %
%% BVA:

waitbar(0,...
    h.Waitbar,...
    'Performing sliding window analysis...');

% Initialize a BurstVarianceAnalysis array:
DATA.Results = nan(TOTALBURSTS, 2);

% Loop through the bursts and perform BVA:
COUNTER = 0;
CLUSTERSTATS = cell(CLUSTERBINS+1, 1);

TRACKER = 0;
for BURST = 1 : TOTALBURSTS

    % Check!!!
    if ismember(BURST, REMOVEBURSTS) % Determine if BURST is blacklisted...
        % Blacklisted burst found! Move along to next burst...
        continue;
    else
        % Update our counter for our non-blacklisted bursts...
        COUNTER = COUNTER + 1;
    end % end IF

    % Grab the current (non-blacklisted) BURST indices:
    INDICES = IndexCell{COUNTER};

    % Remove a 'cutoff' number of photons:
    if numel(INDICES) < PHOTONCUTOFF
        % Go to next iteration...
        REMOVEBURSTS(end+1,1) = BURST;
```

```
        TRACKER = TRACKER + 1;
        continue;
    else
        INDICES(1:PHOTONCUTOFF) = [];
    end % end IF/ELSE


    % Calculate total number of photons:
    TOTALPHOTONS = numel(INDICES);

    % Check!!!
    if TOTALPHOTONS < WINDOW
        % Go to next iteration, as not enough photons in 'burst'...
        REMOVEBURSTS(end+1,1) = BURST;
        TRACKER = TRACKER + 1;
        continue;
    end % end IF



    % We must apply a window to the burst photons; this window is
    % non-overlapping, and thus, in most cases, the number of windows
    % within a burst is not an integer, but rather, a remainder of
    % photons cannot be used.  Let us discard these photons, so we can
    % reshape our photon array:
    RemPhotons = rem(TOTALPHOTONS, WINDOW);

    % We have some leftover photons; remove them...
    INDICES(end-RemPhotons+1:end) = [];


    % Reshape the photons from the arrival matrix into an array of
    % non-overlapping windows: (NOTE: photons within each window are
    % placed along the columns, and the window number within the burst
    % is placed along the rows)
    PhotonArray = transpose(reshape(ArrivalMatrix(INDICES,2), WINDOW, ...
       [])); % No apriori knowledge of how many windows within a burst

    % Calculate the Efficiency value (FRET) within each window: (NOTE:
    % we exploit the fact that DexDem photons are labeled 0):
    EfficiencyVector = sum((PhotonArray == 2), 2) ./ WINDOW; % where ...
       WINDOW = sum of DexDem + DexAem and the numerator is the number of ...
       DexAem photons

    % Calculate the standard deviation of the efficiency values:
    EfficiencyStDev = std(EfficiencyVector);

    % Place the mean and stdev of the efficiency in our BVA array:
    DATA.Results(BURST, :) = [mean(EfficiencyVector), EfficiencyStDev];
```

```
    % Place the efficiency vector with the burst's respective cluster:
    WhichCluster = CLUSTERBINIDX(BURST);

    % Check!!!
    if WhichCluster == (CLUSTERBINS+2)
        % This accounts for values which are precisely equal to the
        % last edge.  We place this data in the last bin.
        WhichCluster = WhichCluster - 1;
    end % end IF/ELSE

    CLUSTERSTATS{WhichCluster} = [CLUSTERSTATS{WhichCluster}; ...
       EfficiencyVector];


    % Update our waitbar:
    waitbar(BURST/TOTALBURSTS, h.Waitbar);


end % end FOR

fprintf(['%d bursts did not have sufficient photons ',...
    'to perform a sliding window calculation.\n\n'],...
    TRACKER);

% Tidy-up our Cluster Statistics:
RemoveClusters = cellfun(@isempty, CLUSTERSTATS);
CLUSTERSTATS(RemoveClusters) = [];
CLUSTEREDGES(RemoveClusters) = [];

RemoveClusters = cell2mat(cellfun(@(x) (numel(x)<CLUSTERMINWINDOWS),...
    CLUSTERSTATS,...
    'UniformOutput', false));
CLUSTERSTATS(RemoveClusters) = [];
CLUSTEREDGES(RemoveClusters) = [];


IDs = setdiff(BURSTINDICES, REMOVEBURSTS); % Retain only 'good' bursts

DATA.Results = DATA.Results(IDs,:);

% Remove any data in which the mean efficiency is either 0 or 1: (why?
% these 'events' tend to arise when having low photon statistics and will
% bunch up at the corner of the plots. Because of discrete photon
% statistics, these will occur often and affect the relatively weighting of
% the data elsewhere which has higher photon statistics and is more
% interesting (usually). Hence, we throw away this data, if present, and
% only keep that data which has more 'realistic' mean values.)
```

```
DATA.Results(DATA.Results(:,1) == 0   DATA.Results(:,1) == 1, :) = [];

% Remove any data in which the standard deviation is zero: (why? Having a
% single STD is not realistic, as fluctuations should be present simply to
% to photon statistics. This also might be indicative of insufficient
% number of WINDOWS allowing for STD calculation. Once again, with low
% photon statistics, this is possible, and can bias the weighting of your
% 2D histogram.)
DATA.Results(DATA.Results(:,2) == 0, :) = [];

% Remove all NaN rows:
DATA.Results(isnan(DATA.Results(:,1)), :) = [];


% Tidy-up:
clear ArrivalMatrix
clear BinIndices

% ----------------------------------------- %
%% Cluster Statistics:

waitbar(0,...
    h.Waitbar,...
    'Analyzing cluster statistics...');


% ------------ %
% [ ] Group bursts with similar mean values and calculate population
% standard deviation:
ClusterCenters = CLUSTEREDGES(1:end-1) + CLUSTERWIDTH/2;

% Initialize some variables:
NumClusters = numel(CLUSTERSTATS);
HighCIvec = zeros(NumClusters, 1);
LowCIvec = zeros(NumClusters, 1);
StDevDistrMean = zeros(NumClusters, 1);

ClusterStats = nan(NumClusters, 2);

% Loop through each 'slice' (or 'cluster' of data) from the 2D Histogram
% and calculate the statistics:
for CLUSTER = 1 : NumClusters

    ClusterStats(CLUSTER,2) = std(CLUSTERSTATS{CLUSTER});
    ClusterStats(CLUSTER,1) = mean(CLUSTERSTATS{CLUSTER});

    % Calculate StDev Distribution:
    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)
```

```
    NumWindows = numel(CLUSTERSTATS{CLUSTER});
    StDevDistr = MonteCarloPrediction(...
        Edges,...
        NumWindows,... % number of windows
        WINDOW,... % number of photons within a window
        ClusterCenters(CLUSTER),... % probability of energy transfer
        NUMMONTECARLOSAMPLES); % number of samples generated by Monte ... Carlo simulation

    CumDistr = cumsum(StDevDistr);
    INV = 1 - CumDistr;
    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

    %-- CI --
    %-- ADJUST CI BY THE  MULT HYPOTH TEST --
    absvec = abs(CumDistr - CONFIDENCEINTERVAL);
    [M, closest] = min(absvec);

    absvecLOW = abs(INV - CONFIDENCEINTERVAL);
    closestLOW = find(absvecLOW == min(absvecLOW),...
        1, 'last');

    % Why are we calculating the CI off an index? The index tells us
    % something of about the number of false positives.
    HighCIvec(CLUSTER,:) = (closest-1)*.001; % 0.001 is 99.9% Confidence ... interval
    LowCIvec(CLUSTER,:) = (closestLOW-1)*.001;

    % Update our waitbar:
    waitbar(CLUSTER / NumClusters,...
        h.Waitbar);


end % end FOR

% Assign over to application data:
DATA.Clusters.Centers = ClusterCenters;
DATA.ConfidenceInterval.Upper = HighCIvec;
DATA.ConfidenceInterval.Lower = LowCIvec;
DATA.Clusters.StDevDistrMean = StDevDistrMean;
DATA.Clusters.Statistics = ClusterStats;
DATA.Clusters.WindowMeans = CLUSTERSTATS;
DATA.Clusters.DistanceFromUpperCI =...
    ClusterStats(:,2) - HighCIvec;


% ------------------------------------------ %
```

```
%% Tidy-up:



% Delete waitbar:
delete(h.Waitbar);


TOTALTIME = toc






return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOF



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% StartUp:
function StartUp()
%
%
%

addpath(...
    'Other/lightspeed',...
    'Other/SplitVec');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CleanUp:
function CleanUp()
%
%
%

rmpath(...
    'Other/lightspeed',...
    'Other/SplitVec');
```

Data Filtering

```
function [DATA, REMOVEBURSTS] = FilterData(...
    ArrivalMatrix, FILTERS)
% FILTERDATA
%   Filter a photon arrival matrix using supplied filter parameters.
%
%   INPUTS:
%       ARRIVALMATRIX: input data on which to perform Burst Variance
%       Analysis. The input matrix should be Nx3 or 3xN, 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 a
%       corresponding index to a unique burst ID; background photons should
%       be indexed as NaN): ARRIVALMATRIX = ArrivalTimes   ID   BurstNumber
%
%       FILTERS: nested 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.
%               (excluding photons detected in donor channel upon acceptor
%               excitation: AexDem)
%           EFFICIENCY: the FRET efficiency.
%           STOICHIOMETRY: the relative fluorophore brightness.
%
%   OUTPUTS:
%       DATA: structure with the following data streams for each detected
%       burst:
%           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.
%               (excluding photons detected in donor channel upon acceptor
%               excitation: AexDem)
%           EFFICIENCY: the FRET efficiency.
%           STOICHIOMETRY: the relative fluorophore brightness.
%
%       REMOVEBURSTS: a vector providing indices for the bursts which
%       failed filter criteria.
%
%   DEPENDENCIES:
%       SPLITVEC.M
%
%   -----------------------
%   EXAMPLE USAGE:
%       [DATA, REMOVEBURSTS] = FilterData(ArrivalMatrix, FILTERS);
%
%   ------------------------
%   History:
%       [1] Kristofer Gryte. University of Oxford (2011).
%
%   Copyright (c) 2011. Kristofer Gryte. University of Oxford.
%
%
%
%   Version: 1.0 (2011-07-30)
%
%   ----------------------
%   Author Information:
%       Kristofer Gryte
%       Clarendon Laboratory
%       University of Oxford
%       Oxford, United Kingdom
%       e-mail: k.gryte1 (at) physics.ox.ac.uk
%
%   ------------------------
%   TODO:
%       [1] Add checks for input variables.
%
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOP


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialization:

```

```
REMOVEBURSTS = [];

Apply = true; % FLAG to determine if apply filters or not. THIS IS NEEDED, ...
    FOR NOW, EVEN IF NO FILTERS ARE APPLIED!!!!

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Assemble Single Data Matrix:

% Replace all the NaNs:
ArrivalMatrix(isnan(ArrivalMatrix(:,3)),3) = 0;

% Return the assigned IDs of the various runs: % recall that non-burst
% photons are labeled with a '0'!!!!
FLAGVALS = SplitVec(...
    ArrivalMatrix(:,3),...ArrivalMatrix(¬isnan(ArrivalMatrix(:,3)),3),... ...
        % exclude any NaNs
    'equal',... % find all those runs of the same index (burst ID)
    'firstval'); % return just the first index marking a run.


% --- %
% [1] Start Times:
STARTINDICES = SplitVec(...
    ArrivalMatrix(:,3),...ArrivalMatrix(¬isnan(ArrivalMatrix(:,3)),3),... ...
        % exclude any NaNs
    'equal',... % find all those runs of the same index (burst ID)
    'first'); % return just the first index marking a run.

STARTINDICES(FLAGVALS == 0) = []; % NOTE:

DATA.StartTimes = ArrivalMatrix(STARTINDICES, 1);

% --- %
% [2] End Times:
ENDINDICES = SplitVec(...
    ArrivalMatrix(:,3),...ArrivalMatrix(¬isnan(ArrivalMatrix(:,3)),3),... ...
        % exclude any NaNs
    'equal',... % find all those runs of the same index (burst ID)
    'last'); % return just the last index completing a run.

ENDINDICES(FLAGVALS == 0) = []; % NOTE:

DATA.EndTimes = ArrivalMatrix(ENDINDICES, 1);

% --- %
% [3] Duration:
DATA.Duration = DATA.EndTimes - DATA.StartTimes;

% --- %
```

```
% [4] Length:
DATA.Length = ENDINDICES - STARTINDICES;

% --- %
BURSTCELL(:,1) = SplitVec(...
    ArrivalMatrix(:, 3),...ArrivalMatrix(¬isnan(ArrivalMatrix(:,3)), 2),...
    'equal',...
    'loc'); % this is a cellular array!

BURSTCELL(FLAGVALS == 0) = []; % NOTE:

% ---------- %
% Free up some memory:
% clear ArrivalMatrix
% ---------- %

% [5] DexDem: % DexDem = 0
DATA.DexDem(:,1) = cellfun(@(x) sum(ArrivalMatrix(x,2) == 0), BURSTCELL, ...
    'UniformOutput', true);

% [6] DexAem: % DexAem = 2
DATA.DexAem(:,1) = cellfun(@(x) sum(ArrivalMatrix(x,2) == 2), BURSTCELL, ...
    'UniformOutput', true);

% [7] AexAem: % AexAem = 1
DATA.AexAem(:,1) = cellfun(@(x) sum(ArrivalMatrix(x,2) == 1), BURSTCELL, ...
    'UniformOutput', true);

% --- %
% [8] Sum Photons: DexDem + DexAem
DATA.SumPhotons = DATA.DexDem + DATA.DexAem;

% --- %
% [9] Total Photons: DexDem + DexAem + AexAem
DATA.TotalPhotons = DATA.SumPhotons + DATA.AexAem;

% --- %
% [10] Efficiency:
DATA.Efficiency = DATA.DexAem ./ (DATA.SumPhotons);

% --- %
% [11] Stoichiometry:
DATA.Stoichiometry = DATA.SumPhotons ./ (DATA.TotalPhotons);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Threshold Data:

if Apply == true
```

```

    %%%%%%%%%%%%%%%%%%%%%%%%%
    % Min Duration:
    if ¬isempty(FILTERS.Duration.Min)

        IDs = find(DATA.Duration < FILTERS.Duration.Min); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Duration(DATA.Duration < FILTERS.Duration.Min,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%
    % Max Duration:
    if ¬isempty(FILTERS.Duration.Max)

        IDs = find(DATA.Duration > FILTERS.Duration.Max); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Duration(DATA.Duration > FILTERS.Duration.Max,:) = NaN;

    end


    %%%%%%%%%%%%%%%%%%%%%%%%%
    % Min Length:
    if ¬isempty(FILTERS.Length.Min)

        IDs = find(DATA.Length < FILTERS.Length.Min); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Length(DATA.Length < FILTERS.Length.Min,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%
    % Max Length:
    if ¬isempty(FILTERS.Length.Max)

        IDs = find(DATA.Length > FILTERS.Length.Max); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Length(DATA.Length > FILTERS.Length.Max,:) = NaN;

    end



    %%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Min DexcDem:
if ¬isempty(FILTERS.DexDem.Min)

    IDs = find(DATA.DexDem < FILTERS.DexDem.Min); % Column vector
    REMOVEBURSTS = [REMOVEBURSTS; IDs];

    DATA.DexDem(DATA.DexDem < FILTERS.DexDem.Min,:) = NaN;

end

%%%%%%%%%%%%%%%%%%%%%%%%%
% Max DexcDem:
if ¬isempty(FILTERS.DexDem.Max)

    IDs = find(DATA.DexDem > FILTERS.DexDem.Max); % Column vector
    REMOVEBURSTS = [REMOVEBURSTS; IDs];

    DATA.DexDem(DATA.DexDem > FILTERS.DexDem.Max,:) = NaN;

end

%%%%%%%%%%%%%%%%%%%%%%%%%
% Min DexcAem:
if ¬isempty(FILTERS.DexAem.Min)

    IDs = find(DATA.DexAem < FILTERS.DexAem.Min); % Column vector
    REMOVEBURSTS = [REMOVEBURSTS; IDs];

    DATA.DexAem(DATA.DexAem < FILTERS.DexAem.Min,:) = NaN;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Max DexcAem:
if ¬isempty(FILTERS.DexAem.Max)

    IDs = find(DATA.DexAem > FILTERS.DexAem.Max); % Column vector
    REMOVEBURSTS = [REMOVEBURSTS; IDs];

    DATA.DexAem(DATA.DexAem > FILTERS.DexAem.Max,:) = NaN;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Min AexcAem:
if ¬isempty(FILTERS.AexAem.Min)

    IDs = find(DATA.AexAem < FILTERS.AexAem.Min); % Column vector
    REMOVEBURSTS = [REMOVEBURSTS; IDs];
```

```
        DATA.AexAem(DATA.AexAem < FILTERS.AexAem.Min,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Max AexcAem:
    if ¬isempty(FILTERS.AexAem.Max)

        IDs = find(DATA.AexAem > FILTERS.AexAem.Max); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.AexAem(DATA.AexAem > FILTERS.AexAem.Max,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Min Sum DD+DA:
    if ¬isempty(FILTERS.SumPhotons.Min)

        IDs = find(DATA.SumPhotons < FILTERS.SumPhotons.Min); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.SumPhotons(DATA.SumPhotons < FILTERS.SumPhotons.Min,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Max Sum DD+DA:
    if ¬isempty(FILTERS.SumPhotons.Max)

        IDs = find(DATA.SumPhotons > FILTERS.SumPhotons.Max); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.SumPhotons(DATA.SumPhotons > FILTERS.SumPhotons.Max,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Min Total Signal:
    if ¬isempty(FILTERS.TotalPhotons.Min)

        IDs = find(DATA.TotalPhotons < FILTERS.TotalPhotons.Min); % Column ...
            vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.TotalPhotons(DATA.TotalPhotons < FILTERS.TotalPhotons.Min,:) ...
            = NaN;

```

```
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Max Total Signal:
    if ¬isempty(FILTERS.TotalPhotons.Max)

        IDs = find(DATA.TotalPhotons > FILTERS.TotalPhotons.Max); % Column ...
            vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.TotalPhotons(DATA.TotalPhotons > FILTERS.TotalPhotons.Max,:) ...
            = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Min Efficiency:
    if ¬isempty(FILTERS.Efficiency.Min)

        IDs = find(DATA.Efficiency < FILTERS.Efficiency.Min); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Efficiency(DATA.Efficiency < FILTERS.Efficiency.Min,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Max Efficiency:
    if ¬isempty(FILTERS.Efficiency.Max)

        IDs = find(DATA.Efficiency > FILTERS.Efficiency.Max); % Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Efficiency(DATA.Efficiency > FILTERS.Efficiency.Max,:) = NaN;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Min Stoichiometry:
    if ¬isempty(FILTERS.Stoichiometry.Min)

        IDs = find(DATA.Stoichiometry < FILTERS.Stoichiometry.Min); % ...
            Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Stoichiometry(DATA.Stoichiometry < ...
            FILTERS.Stoichiometry.Min,:) = NaN;

    end
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Max Stoichiometry:
    if ¬isempty(FILTERS.Stoichiometry.Max)

        IDs = find(DATA.Stoichiometry > FILTERS.Stoichiometry.Max); % ...
            Column vector
        REMOVEBURSTS = [REMOVEBURSTS; IDs];

        DATA.Stoichiometry(DATA.Stoichiometry > ...
            FILTERS.Stoichiometry.Max,:) = NaN;

    end

end % end IF


% Retain only the unique indices:
REMOVEBURSTS = unique(REMOVEBURSTS);




% FOR NOW...
return;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOF
```

Monte Carlo Simulation

```
function STDDISTRIBUTION = MonteCarloPrediction(EDGES, EVENTS,...
    TRIALS, PROBABILITY, ITERATIONS)
% MONTECARLOPREDICTION
%   Use Monte Carlo methods to generate a distribution of standard
%   deviation (std) values observed from a binomial process (Bernoulli
%   Trial) with a specified probability of success.
%
%   INPUTS:
%       EDGES: a vector defining the bin edges into which the simulated
%       standard deviations will be histogrammed.
%
%       EVENTS: the number of events (collections of trials) to be
%       sampled from the distribution.
%
%       TRIALS: the number of trials.
%
%       PROBABILITY: the probability of success. For example, a series of
%       coin flips is a collection of events characterized by Bernoulli
%       trials with probability 0.5.
%
%       ITERATIONS: the number of times to repeat each collection of events
%       so as to arrive at an average value for the calculated standard
%       deviations. Default: 500.
%
%
%   OUTPUTS:
%       STDDISTRIBUTION: a probability distribution of the standard
%       deviation with support defined by EDGES. Output is a vector.
%
%   DEPENDENCIES:
%       BINORND.M (STATS toolbox)
%
%   --------------------------
%   EXAMPLE USAGE:
%       MonteCarloPrediction([0:0.0001:1], 5, 100, 0.5);
%
%       MonteCarloPrediction([0:0.0001:1], 5, 100, 0.5, 500);
%
%   --------------------------
%   History:
%       [1] Joseph Torella, Yusdi Santoso. University of Oxford (2009).
%       [2] Kristofer Gryte. University of Oxford (2011).
%
%   Copyright (c) 2011. Kristofer Gryte. University of Oxford.
%
```

```
%
%
%   Version: 1.0 (2011-07-30)
%
%   ----------------------
%   Author Information:
%       Kristofer Gryte
%       Clarendon Laboratory
%       University of Oxford
%       Oxford, United Kingdom
%       e-mail: k.gryte1 (at) physics.ox.ac.uk
%
%   ---------------------------
%   TODO:
%       [1] Generate a function independent of the STATS toolbox.
%
%
%   ---------------------------
%   See also BINORND
%
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOP





%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Check Input Arguments:

% Check!
if nargin ≤ 4
    ITERATIONS = 500; % default value
end % end IF

% Check EDGES:
if size(EDGES,1) ≠ 1 && size(EDGES,2) ≠ 1
    % Throw an error:
    errordlg(['ERROR:invalid input argument. Please input a vector for the ...
        ',...
        'EDGES argument.'],...
        'ERROR:invalid input argument');

    return;
end % end IF

% Check EVENTS, TRIALS, PROBABILITY, and ITERATIONS
```

```
temp = [EVENTS, TRIALS, ITERATIONS];
if sum((¬isinteger(temp)) + (temp>0)) ≠ 6
    % Throw an error:
    errordlg(['ERROR:invalid input argument. Please input positive ...
        integers for the ',...
        'EVENTS, TRIALS, and ITERATIONS arguments.'],...
        'ERROR:invalid input argument');

    return;
end % end IF

if (PROBABILITY ≤ 0    PROBABILITY > 1)
    % Throw an error:
    errordlg(['ERROR:invalid input argument. Please input a probability ...
        between 0 and 1 ',...
        'for the PROBABILITY argument.'],...
        'ERROR:invalid input argument');

    return;
end % end IF




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialization:

% --- %
% Parameters:
Runs = 3; % used in the event of a large number of requested standard ...
    deviations, in which case, for memory purposes, we need to break the ...
    work into smaller chunks.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generate Monte Carlo Prediction:

if EVENTS*ITERATIONS ≤ 10000 % 10,000 ? A magical number discovered ...
    heuristically.

    % Generate the matrix of numbers drawn from a binomial distribution
    % with total trials equal to TRIALS and probability equal to
    % PROBABILITY:
    MyMatrix = binornd(...
        TRIALS,...
        PROBABILITY,...
        EVENTS,...
        ITERATIONS); % this is a matrix of size: EVENTS x ITERATIONS
```

```

    % Calculate the standard deviation along each column
    StdMatrix = std(MyMatrix ./ TRIALS);

elseif EVENTS*ITERATIONS > 10000

    % With many iterations, we run into the problem of very large matrices
    % and memory issues. Let us break it down more manageably:
    % ---- %

    RemainingSamples = ITERATIONS;

    % Pre-allocate our array:
    StdMatrix = nan(1, ITERATIONS);

    % Initialize an index counter:
    Counter = 1;

    for k = 1 : Runs

        % Determine the number of samples to run:
        Samples = floor(RemainingSamples / (Runs-k+1));

        % Generate the matrix of numbers drawn from a binomial distribution
        % with total trials equal to TRIALS and probability equal to
        % PROBABILITY:
        MyMatrix = binornd(...
            TRIALS,...
            PROBABILITY,...
            EVENTS,...
            Samples);

        % Calculate the standard deviation along each column and append to
        % our standard deviation matrix:
        StdMatrix(Counter : (Counter+Samples-1)) = std(MyMatrix ./ TRIALS);

        % Calculate the remaining number of samples:
        RemainingSamples = RemainingSamples - Samples;

        % Update our counter:
        Counter = Counter + Samples;


    end % end FOR


end % end IF/ELSEIF

% Histogram the standard deviations where the bin edges are defined by
```

```
% EDGES:
[Counts, BinIndices] = histc(StdMatrix, EDGES);

% Determine the probability of a particular standard deviation, by
% normalizing by the total number of counts:
STDDISTRIBUTION = Counts / sum(Counts); % this is a vector


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOF
```

Visualization

```
function [DATA, handles] = PlotResults(DATA, WINDOW, handles)
% PLOTRESULTS
%   Plots the DATA results generated by Burst Variance Analysis into the
%   axes supplied by HANDLES. Configuration and aesthetics of the axes is
%   currently controlled internally. NOTE: the BVA data is smoothed from a
%   discretized histogram presentation and transformed to a continuous
%   contour. The reference for this smoothing is " Garcia D, Robust
%   smoothing of gridded data in one and higher dimensions with missing
%   values. Computational Statistics & Data Analysis, 2010. "
%
%   INPUTS:
%       DATA: structure with the following data streams for each detected
%       burst: (NOTE: data streams are filtered according to input FILTERS)
%           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.
%               (excluding photons detected in donor channel upon acceptor
%               excitation: AexDem)
%           EFFICIENCY: the FRET efficiency.
%           STOICHIOMETRY: the relative fluorophore brightness.
%           RESULTS: an Nx2 array, where N is the number of bursts (after
%               filtering). The first column corresponds to the mean ...
    Efficiency
%               value for a burst and the second to the burst's Efficiency
%               standard deviation. Array: E(efficiency)   sqrt(V(efficiency))
%           CLUSTERS: nested structure with the following fields:
%               CENTERS: vector containing the centers of the individual
%                  clusters.
%               STDEVDISTRMEAN: the Monte Carlo simulated standard
%                  deviation prediction.
%               STATISTICS: an Mx2 array, where M is the number of clusters
%                   ('slices' along the efficiency axis, into which bursts are
%                   grouped). The first column corresponds to the mean
%                   Efficiency value for the cluster, and the second to the
%                   standard deviation of the burst's within that cluster.
%           CONFIDENCEINTERVAL: nested structure with the following fields:
```

```
%               UPPER: the upper confidence bound generated from the Monte
%                  Carlo simulation and multiple hypothesis test.
%               LOWER: the lower confidence bound generated from the Monte
%                  Carlo simulation and the multiple hypothesis test.
%
%       WINDOW: size of the sliding window over which FRET will be
%       calculated. Default: 5.
%
%       HANDLES: structure with the following fields:
%           FIGURE: handle to the figure window.
%           SUBPLOT: nested structure with the following fields:
%               BVA: nested structure with the following fields:
%                   AXES: handles to the 2D Histogram axes (subplot).
%           SUBPLOT: nested structure with the following fields:
%               HISTOGRAM: nested structure with the following fields:
%                   MEAN: nested structure with the following fields:
%                      AXES: handle to the axes into which the histogram
%                         will be plotted.
%                   STDEV: nested structure with the following fields:
%                      AXES: handle to the axes into which the histogram
%                         will be plotted.
%
%
%
%   OUTPUTS:
%       DATA: input structure with the following added data streams:
%           HISTOGRAMS: nested structure with the following fields:
%               COUNTS: the counts per bin for the 2D histogram.
%               EDGES: the edges used to define the histogram bins.
%               BINCENTERS: the locations of the center of each bin.
%               SMOOTHED: the histogram data after application of a
%                  smoothing filter.
%
%       HANDLES: input structure with the following added fields:
%           PLOTS: nested structure with the following fields:
%               CONTOUR: handle to the counter plot created from the 2D
%                   histogram
%               SHOTNOISEPREDICTION: handle to the shot noise prediction
%                  based on the number of photons specified in WINDOW.
%               CONFIDENCEINTERVAL: nested structure with the following
%                  fields:
%                   UPPER: handle to the upper confidence interval
%                      generated via a Monte Carlo simulation. Cluster STD
%                      values above this limit are indicative of dynamics.
%                   MEAN: handle to the average of upper and lower
%                      confidence bounds. This should correspond to the
%                      SHOTNOISEPREDICTION curve.
%               STATIC: STD clusters which fall below the upper confidence
%                  interval are considered 'static'. This is the handle to
```

```
%                  these data points.
%               DYNAMICS: STD clusters which are above the upper confidence
%                  interval are considered 'dynamic'. This is the handle to
%                  these data points.
%               UNKNOWN: STD clusters which fall below the lower confidence
%                  interval are categorized as 'unknown'. The causes of this
%                  behavior are not well understood.
%           SUBPLOT: nested structure with the following fields:
%               HISTOGRAM: nested structure with the following fields:
%                   MEAN: nested structure with the following fields:
%                       PLOTS: nested structure with the following fields:
%                           BAR: handle to the Mean bar graph data.
%                   STDEV: nested structure with the following fields:
%                       PLOTS: nested structure with the following fields:
%                           BAR: handles to the StDev bar graph data.
%           COLORBAR: handle to the colobar corresponding to the 2D
%              histogram data.
%
%
%
%   DEPENDENCIES:
%       SMOOTHN.M
%       HISTCN.M
%
%   -----------------------
%   EXAMPLE USAGE:
%       [DATA, handles] = PlotResults(DATA, WINDOW, handles);
%
%   ------------------------
%   History:
%       [1] Kristofer Gryte. University of Oxford (2011).
%
%   Copyright (c) 2011. Kristofer Gryte. University of Oxford.
%
%
%
%   Version: 1.0 (2011-08-02)
%
%   ----------------------
%   Author Information:
%       Kristofer Gryte
%       Clarendon Laboratory
%       University of Oxford
%       Oxford, United Kingdom
%       e-mail: k.gryte1 (at) physics.ox.ac.uk
%
%   ------------------------
%   TODO:
%       [1] Allow axes configurations to be an input structure.
```

```
%       [2] Provide checks on input arguments.
%       [3] Figure out why 'histcn' appears to reset the path(s).
%
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOP


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% StartUp/CleanUp:

StartUp();

C = onCleanup(@() CleanUp());


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialization:

% --- %
% Check!!!
if nargin ≠ 3
    % Throw an error:
    errordlg(['ERROR:not enough input arguments. Please provide the ... following ',...
        'input arguments: DATA, WINDOW, HANDLES. See the HELP for the more ... ',...
        'information.'],...
        'ERROR:not enough input arguments');
    return;
end % end IF

% --- %
% Default Configurations:

Config.Histogram2D.Smoothing = 20; % Smoothing parameter

Config.ShotNoise.Prediction.OrdinateVals = 0:0.01:1; % Efficiency sampling....
Config.ShotNoise.Prediction.StDev = sqrt(...
    (Config.ShotNoise.Prediction.OrdinateVals .* ... (1-Config.ShotNoise.Prediction.OrdinateVals))...
    /...
    WINDOW);

Config.Axes.XLabel.FontSize = 14;
Config.Axes.XLabel.String = 'Mean';

Config.Axes.YLabel.FontSize = 14;
```

```
Config.Axes.YLabel.String = 'StDev';

Config.Axes.Title.FontSize = 14;
Config.Axes.Title.String = 'Burst Variance Analysis (Efficiency)';

Config.Axes.TickDir = 'out';
Config.Axes.Box = 'off';

Config.Figure.BackgroundColor = 'w';

Config.Legend.Box = 'off';
Config.Legend.Location = 'south';


Config.Histogram.Mean.Axes.YLabel.FontSize = 14;
Config.Histogram.Mean.Axes.YLabel.String = 'Counts';

Config.Histogram.Mean.Axes.Box = 'off';
Config.Histogram.Mean.Axes.TickDir = 'out';


Config.Histogram.StDev.Axes.XLabel.FontSize = 14;
Config.Histogram.StDev.Axes.XLabel.String = 'Counts';

Config.Histogram.StDev.Axes.Box = 'off';
Config.Histogram.StDev.Axes.TickDir = 'out';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% ---------- %
%% Plot:

% Activate our figure:
figure(handles.figure);

% Activate our BVA Axes:
subplot(handles.Subplot.BVA.Axes);

% Get the histogram counts: (NOTE: default is 100 Bins)
[DATA.Histogram.Counts, DATA.Histogram.Edges, DATA.Histogram.BinCenters] = ...
    histcn(...
     DATA.Results);

StartUp(); %%% NOTE!!! For some reason, 'histc'/'histcn' appears to reset ...
    the path(s)

% 'Smooth' the histogram:
DATA.Histogram.Smoothed = smoothn(...
```

```
    DATA.Histogram.Counts,...
    Config.Histogram2D.Smoothing);

% ------------ %
% [1] Create the contour plot of the burst standard deviations:
[ContourMatrix, handles.Plots.Contour] = contourf(...
    DATA.Histogram.BinCenters{1},...
    DATA.Histogram.BinCenters{2},...
    DATA.Histogram.Smoothed(:,:,1)',...
    10,...
    'EdgeColor','none',...
    'tag', 'Per Burst StDev');

hold(handles.Subplot.BVA.Axes, 'on');

% ------------ %
% [2] Display the shot noise prediction:
handles.Plots.ShotNoisePrediction = plot(...
    Config.ShotNoise.Prediction.OrdinateVals,...
    Config.ShotNoise.Prediction.StDev,...
    'k-',...
    'LineWidth', 3.0,...
    'tag', 'Shot Noise Prediction');

% ---------------------- %
% [3] Plot the confidence bounds:
handles.Plots.ConfidenceInterval.Upper = plot(...
    DATA.Clusters.Centers,...
    DATA.ConfidenceInterval.Upper,...
    'm-.',...
    'LineWidth', 2.5,...
    'tag', 'Upper CI');

handles.Plots.ConfidenceInterval.Mean = plot(...
    DATA.Clusters.Centers(:, 1),...
    DATA.Clusters.StDevDistrMean(:, 1),...
    'y--',...
    'LineWidth', 2.0,...
    'MarkerSize', 10,...
    'tag', 'Mean MC Prediction');

% --------------------- %
% [4] Determine where the clustered populations reside relative to the
% confidence bounds:
LogID = (DATA.Clusters.Statistics(:,2) ≤ DATA.ConfidenceInterval.Upper...
     ...
    DATA.Clusters.Statistics(:,2) ≥ DATA.ConfidenceInterval.Lower);

handles.Plots.Static = plot(...
```

```
    DATA.Clusters.Centers(LogID, 1),...
    DATA.Clusters.Statistics(LogID, 2),...
    'k^',...
    'LineWidth', 1.5,...
    'MarkerFaceColor','k',...
    'MarkerSize', 8,...
    'tag', 'Static');


LogID = (DATA.Clusters.Statistics(:,2) > DATA.ConfidenceInterval.Upper);
handles.Plots.Dynamics = plot(...
    DATA.Clusters.Centers(LogID, 1),...
    DATA.Clusters.Statistics(LogID, 2),...
    'k^',...
    'LineWidth', 1.5,...
    'MarkerFaceColor','r',...
    'MarkerSize', 8,...
    'tag', 'Dynamics');

LogID = (DATA.Clusters.Statistics(:,2) < DATA.ConfidenceInterval.Lower);
handles.Plots.Unknown = plot(...
    DATA.Clusters.Centers(LogID, 1),...
    DATA.Clusters.Statistics(LogID, 2),...
    'k^',...
    'LineWidth', 1.5,...
    'MarkerFaceColor','g',...
    'MarkerSize', 8,...
    'tag', 'Unknown');



hold(handles.Subplot.BVA.Axes, 'off');

% --------------------- %
% [6] Compute Histograms:

DATA.Histogram.Edges{1,1} = -0.025 : 0.05 : 1.025; % Comment out to retain ...
   3D hist edges

Counts1 = histc(DATA.Results(:,1),...
    DATA.Histogram.Edges{1,1});

handles.Subplot.Histogram.Mean.Plots.Bar =...
    bar(handles.Subplot.Histogram.Mean.Axes,...
        DATA.Histogram.Edges{1,1}, Counts1, 'histc');

Counts2 = histc(DATA.Results(:,2),...
    DATA.Histogram.Edges{1,2});

```

```
handles.Subplot.Histogram.StDev.Plots.Bar =...
    barh(handles.Subplot.Histogram.StDev.Axes,...
        DATA.Histogram.Edges{1,2}, Counts2, 'histc');

% --------------------- %
%% Adjust Axes:

% BVA:
xlim(handles.Subplot.BVA.Axes,...
    [min(Config.ShotNoise.Prediction.OrdinateVals),...
    max(Config.ShotNoise.Prediction.OrdinateVals)]);

ylim(handles.Subplot.BVA.Axes,...
    [0, max(DATA.Results(:,2))]);

set(handles.figure,...
    'color', Config.Figure.BackgroundColor)

ylabel(handles.Subplot.BVA.Axes,...
    Config.Axes.YLabel.String,...
    'FontSize', Config.Axes.YLabel.FontSize);

xlabel(handles.Subplot.BVA.Axes,...
    Config.Axes.XLabel.String,...
    'FontSize', Config.Axes.XLabel.FontSize)

title(handles.Subplot.BVA.Axes,...
    Config.Axes.Title.String,...
    'FontSize', Config.Axes.Title.FontSize);

set(handles.Subplot.BVA.Axes,...
    'TickDir', Config.Axes.TickDir,...
    'Box', Config.Axes.Box);


% Histograms:
xlim(handles.Subplot.Histogram.Mean.Axes,...
    [min(Config.ShotNoise.Prediction.OrdinateVals),...
    max(Config.ShotNoise.Prediction.OrdinateVals)]);

ylabel(handles.Subplot.Histogram.Mean.Axes,...
    Config.Histogram.Mean.Axes.YLabel.String,...
    'FontSize', Config.Histogram.Mean.Axes.YLabel.FontSize);

set(handles.Subplot.Histogram.Mean.Axes,...
    'Box', Config.Histogram.Mean.Axes.Box,...
    'TickDir', Config.Histogram.Mean.Axes.TickDir);
```

```
ylim(handles.Subplot.Histogram.StDev.Axes,...
    [0, max(DATA.Results(:,2))]);

xlabel(handles.Subplot.Histogram.StDev.Axes,...
    Config.Histogram.StDev.Axes.XLabel.String,...
    'FontSize', Config.Histogram.StDev.Axes.XLabel.FontSize);

set(handles.Subplot.Histogram.StDev.Axes,...
    'Box', Config.Histogram.StDev.Axes.Box,...
    'TickDir', Config.Histogram.StDev.Axes.TickDir);


% ----------------------- %
% Create a colorbar:

APOS = get(handles.Subplot.BVA.Axes, 'position');

handles.Colorbar = colorbar('peer', handles.Subplot.BVA.Axes);

CPOS = get(handles.Colorbar, 'position');

set(handles.Colorbar,...
    'Position', [APOS(1)+APOS(3)+0.01, CPOS(2), CPOS(3), CPOS(4)]);

% ----------------------- %
% Configure the Legend:
% handles.Plots.Current = [handles.Plots.Contour, ... handles.Plots.ShotNoisePrediction,...
%     handles.Plots.ConfidenceInterval.Upper, ... handles.Plots.ConfidenceInterval.Lower,...
%     handles.Plots.ConfidenceInterval.Mean,...
%     handles.Plots.Static, handles.Plots.Dynamics, handles.Plots.Unknown];

% handles.Legend = legend(...
%     handles.Plots.Current,...
%     get(handles.Plots.Current, 'tag'));
%
% set(handles.Legend,...
%     'Box', DATA.BVA.Config.Legend.Box,...
%     'Location', DATA.BVA.Config.Legend.Location);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EOF



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% StartUp:
function StartUp()
%
%
%

% Add path(s):
addpath(...
    'Other/histcn',...
    'Other/smoothn');




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CleanUp:
function CleanUp()
%
%
%

% Remove path(s):
rmpath(...
    'Other/histcn',...
    'Other/smoothn');
```