**## Anaconda - Python - Jupyter Environment**

-Requires Anaconda Navigator Installed

1. Open a Terminal (MacOS) or Prompt window (Windows)

2. Create a New CONDA environment with Python 3.8

(base) USER o ~ % conda create --name tttrlib\_env python=3.8

3. Activate environment

% conda activate tttrlib\_env

4. Install TTTRlib

% pip install tttrlib

5. Install other dependencies

% pip install tqdm

% pip install pandas

% pip install scipy

% pip install seaborn

6. Restart Anaconda Navigator

7. Select Environment

tttrlib\_env

8. Open Jupyter Notebook

## Running BVA

1. In Cell # 11, Select the path for files and BIDs

2. In Cell # 15 Modify Correction Factors accordingly

3. In Cell # 27, change the number of photons per Analysis

(Example) number\_of\_photons\_per\_slice = 5 # Change this every run

4. Run entire Notebook

**Script Description:**

**1.**The script inside notebook folder can be used to compute BVA (Beam-View Anisotropy) histogram. The script requires Conda environment with Python 3.8 in tttrlib environment (tttrlib\_env=0.23.9). Other dependencies include tqdm, pandas, scipy and seaborn libraries. The git hub(<https://github.com/SMB-Lab/feda_tools.git>) cloning can be followed by importing other dependencies.

**2.Customise the Function:**

At first, we define a function “compute\_bva\_anisotropy\_mean\_std” which designed for analyzing data related to photon detection, specifically focusing on green parallel and perpendicular channels. It calculates statistical measures (mean and standard deviation) of anisotropy and photon counts within certain time windows.

This description of the function focuses on selecting and processing time-tagged time-resolved (TTTR) data from the burst defined by the rows in the Data Frame. Here ‘ff’ and ‘fl’ are indices or identifiers for the first and last files containing TTTR data. The ‘tttr” selects the TTTR data object for the ‘ff’ file from the ‘tttrs’ dictionary. The “time\_calibration” retrieves the global time resolution from the TTTR data header. This is used to calibrate time measurements.

Splitting by Photon Count:Then it divides the burst data into chunks, each containing “number\_of\_photons\_per\_slice photons”.Which helps to creates a list of TTTR objects, each representing a chunk of data with the specified number of photons.

To compute the proximity ratio for each time window in a burst, while also handling the background and correction factors, our code processes TTTR data, masks, and computes various statistics.

\* Selecting Data: Next the script retrieves and calibrates TTTR data based on file indices and time resolution. Then extracts TTTR events within the defined burst.

**\***Time Window vs. Photon Count: Depending on whether number\_of\_photons\_per\_slice is specified, the burst data is either divided into time windows or chunks of a specified number of photons.

This processing prepares the TTTR data for subsequent analysis, which likely involves calculating the mean and standard deviation for anisotropy and photon counts as indicated in the initial part of the function.

#In this segment of the ‘compute\_bva\_anisotropy\_mean\_std function’, the code processes each time window (TW) within a burst to compute various metrics related to photon counts and anisotropy.

\*Photon For each time window, the code filters and counts the number of photons in the green parallel and perpendicular channels.

\*Microtime Calculation: Computes the mean micro time for these photons.

#Convert and Correct: Converts photon counts to arrays and calculates correction factors and background adjustments.

The next steps in the function likely involve using these results to compute the mean and standard deviation of the anisotropy ratios, based on the n\_green\_p, n\_green\_s, mt\_green\_p, and mt\_green\_s values along with the correction factors.

Finally,

* **Anisotropy Ratio Calculation**: Computes the anisotropy ratios for each time window and applies correction factors.
* **Statistical Measures**: Calculates and stores the mean and standard deviation of anisotropy ratios and microtime measurements.
* **DataFrame Update**: Updates the DataFrame with the computed anisotropy statistics.

This completes the processing and analysis, providing statistical measures of anisotropy and microtime distributions based on the provided TTTR data.

**3.The “average\_histogram”** function computes the mean and standard deviation of values over bins in a 2D histogram. The function calculates the mean and standard deviation for each bin in a 2D histogram, where counts represent the histogram counts, and x\_bins and y\_bins define the bin edges for the x and y dimensions, respectively.

In this function we get,

* **Mean Calculation**: For each x-bin, it calculates the weighted mean of y-values using histogram counts.
* **Standard Deviation Calculation**: Computes the standard deviation based on the weighted variance of y-values.
* **Output**: Returns two arrays—one for the means and one for the standard deviations for each bin along the x-axis.

This function is useful for analyzing 2D histograms where you want to compute statistical measures across the bins, especially in data analysis and visualization contexts.

**4. Next we provide our PTU** data file to the program and use a function ‘ filter\_burstids’ from a module or package named an to filter a DataFrame based on burst IDs specified in a directory or file path bid\_path. For this we provide all the necessary correction factors, background values and G-values required for relative efficiency. In the code:

* **B\_p and B\_s**: represents Background noise rates for parallel and perpendicular channels.
* **l1\_japan\_corr and l2\_japan\_corr** represents correction factors to adjust for systematic errors.

These parameters help in obtaining accurate and meaningful results from photon counting experiments by correcting noise and biases.

**5. Next we go to plot various Burstwise MFD**.

We try to create histogram over burst durations, Count Rate & Number of Photons.

* We create a histogram to visualize the distribution of burst durations.
* It calculates the mean duration of bursts from the histogram data and computes the minimum window length.

\*Next histogram aims to plot a histogram of the Tau (green) column from our Data Frame, converting the column to a float if it was not already in numeric format. This code will handle non-numeric entries and plot the histogram as expected.

\*Next code is for plotting a 2D histogram of Tau (green) versus r Scatter (green) {Anisotropy Vs Time} and saving it to a PDF file.

# Further, we calculate mean burst duration, count rate & no. of photons: The provided code calculates various statistics related to burst duration and count rates from histogram data.

* **Mean Duration**: Calculates the average duration of bursts from the histogram data.
* **Mean Count Rate**: Calculates the average count rate from the histogram data.
* **Duration Time Window**: Determines how long each time window should be to accommodate the specified number of photons per slice based on the average count rate. Here in each run, we change the no. of photons per slice.

**6. Here, we compute mean and variances in bursts**:

The code provided computes BVA (Beam-View Anisotropy) anisotropy statistics and creates a 2D histogram plot of the results.

This function calculates the mean and standard deviation of the anisotropy for each burst in the Data Frame. The number\_of\_photons\_per\_slice parameter specifies the number of photons to use per slice, which influences the time window calculations.

**Function**: The function ‘compute\_bva\_anisotropy\_mean\_std’ is expected to compute anisotropy values and update the DataFrame ‘df’with new columns for anisotropy mean and standard deviation.

**2D Histogram**: The plot visualizes the distribution of anisotropy mean versus the squared anisotropy standard deviation.

**7.Making final plot**:

It computes and plot the excess variance of anisotropy against its mean using a 2D histogram. Where **x** = Mean anisotropy values from the Data Frame and **y** = Excess variance of anisotropy calculated and displayed there.

**8.The next two parts save histogram data into a CSV file.**

H**istogram Data Extraction**: Collects the edges/width and counts from the histogram. Then formats the collected data into a Data Frame with appropriate column names. Then finally the outputs the Data Frame to a CSV file named according to the number of photons per slice.

**9. Finally Steps to Save Two-Dimensional Histogram**

1. **Convert the 2D Histogram Data to a Data Frame**:
   * It extracts the necessary information from hist\_2d, which usually contains counts and bin edges for both dimensions. It constructs the filename dynamically and uses the to csv method to save the Data Frame.