Examples of Applications of the COM-Poisson code

Daniel Durnford

October 26, 2018

This document contains several examples/demonstrations of applications of the COM-Poisson code. These can be useful to confirm that the code and look-up tables are set up properly, run diagnostic tests, and showcase the various utilities of this code. The code to perform the demonstrations is in the file Examples.cc, which contains an exact copy of the main functions of COMPoisson.cc. Before proceeding, prepare the code following the same procedure as with COMPoisson.cc (see the README file). Examples.cc contains a function to perform each demonstration (Example1(), Example2(), etc.). These can be copied into COMPoisson.cc, so they can be used as trouble-shooting tools for that code as well. To run each demonstration, simply call the appropriate function in main() of Examples.cc. The examples are described below.

1 Drawing random numbers from the COM-Poisson distribution

The point of this example is to build a distribution with a desired mean μ and Fano factor F (using COM-Poisson, Bernoulli, Poisson, or a Gaussian where appropriate), draw random numbers from that distribution, and plot them in a histogram. This simple function can serve as a good first test to confirm that the code is set up properly. The code to execute this example is contained in the function Example1(mu, F). Simply call this function in main() with the desired mean and Fano factor as inputs, re-compile Examples.cc, and run Examples.exe. Running the code with the line Example1(10., 0.2); should produce the histogram shown in Fig. 1. The function Example1() works in the following way:

1. First, the look-up tables are loaded by calling the function TableReader().
2. The function BuildCOM(mu, F, verbose) is called to build a probability distribution for the specified mean and Fano factor. This function decides which method is appropriate (i.e. Bernoulli, COM-Poisson, etc.), then creates and stores its cumulative distribution function in the global std :: vector "cdf".
3. The function cdfDraw(draw) is used to draw random numbers from the distribution.
4. A histogram of the random numbers is created.

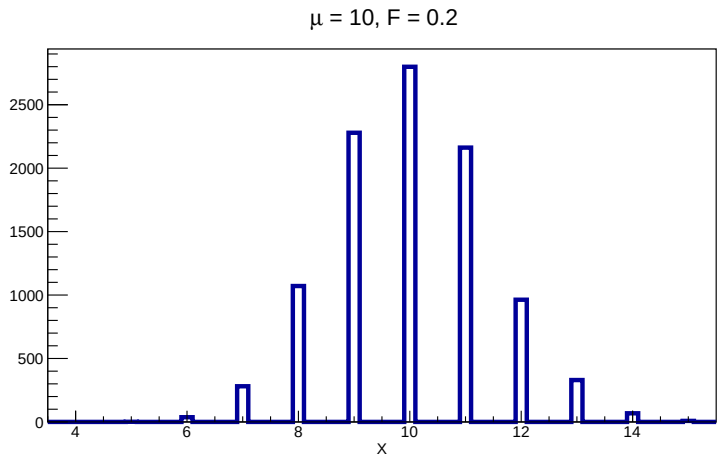

Figure 1: Example of a histogram of random numbers drawn from the COM-Poisson distribution with $\mu = 10$ and $F = 0.2$, made with the function Example1().

2 Speed test comparing fast and slow versions of the code

The functions BuildCOM() and PCOM() (which returns the probability of getting x with a given μ and F, see "Function Summary") were not written carelessly with regards to computational efficiency, but they can at times be somewhat slow. This is because they perform a series of checks for various contingencies and comparisons between the various methods to deliver the most accurate results possible. Faster versions of these two functions (BuildCOM_Fast() and PCOM_Fast()) were created without all the "bells and whistles", potentially sacrificing a marginal amount of accuracy for faster computation times. This example demonstrates the relative computation speed of the two versions of BuildCOM(). The code to execute this example is contained in the function Example2(). Call this function in main(), re-compile Examples.cc, and run Examples.exe, which should produce the plot shown in Fig. 2. Note that using your computer for other tasks at the same time could bias the results of the test. The function Example2() works in the following way:

1. First, the look-up tables are loaded by calling the function TableReader().
2. Then in a for-loop, the functions BuildCOM(mu, F, verbose) and BuildCOM_Fast(mu, F) are called M times each for logarithmically increasing values of μ (and F = 0.2 by default).
3. The time taken at each value of μ is recorded and plotted as a function of μ for both versions of the code, as shown in Fig. 2.

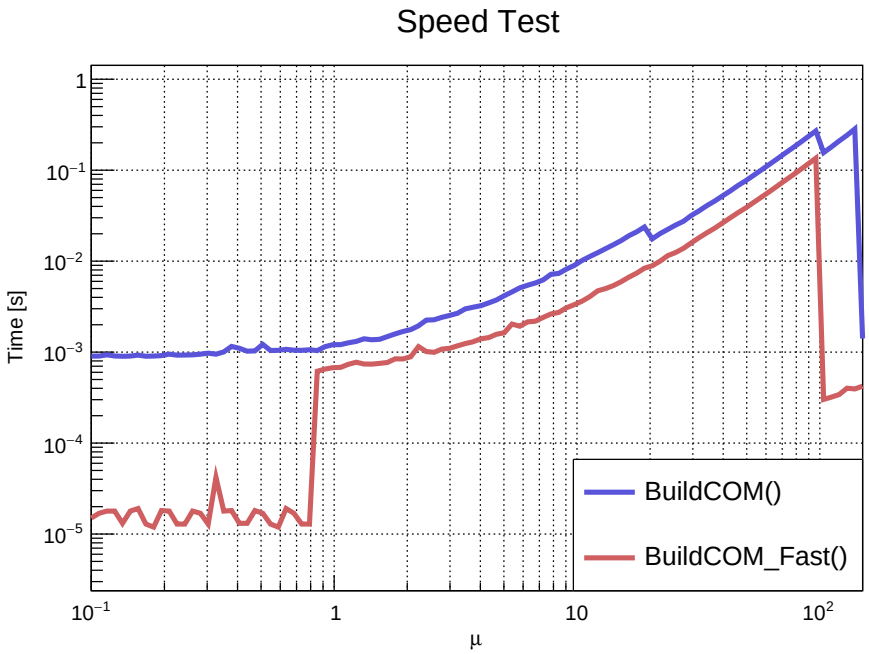

Figure 2: A speed test comparing BuildCOM() and BuildCOM_Fast() as a function of μ, made with the function Example2().

The features visible are due to the four regimes defined in the BuildCOM() and PCOM() codes: the Bernoulli regime, look-up table regime, asymptotic regime, and Gaussian regime (from low μ to high μ). The Bernoulli and Gaussian regimes are fast with BuildCOM_Fast() because they use analytical functions, but are slow with BuildCOM() because of the series of choices (if-statements) considered before using them.

3 Performing an evaluation of the accuracy of the look-up tables

This example allows you to perform an evaluation of the accuracy of the look-up tables, similar to Fig. 6 of [1]. This involves drawing N random points ($N = 10^5$ by default) in (μ, F) parameter space, using the appropriate method to build a distribution, calculating the errors of the obtained mean and Fano factor relative to the desired values, and plotting these values in histograms. The code to execute this example is contained in the function Example3(). Call this function in main(), re-compile Examples.cc, and run Examples.exe, which

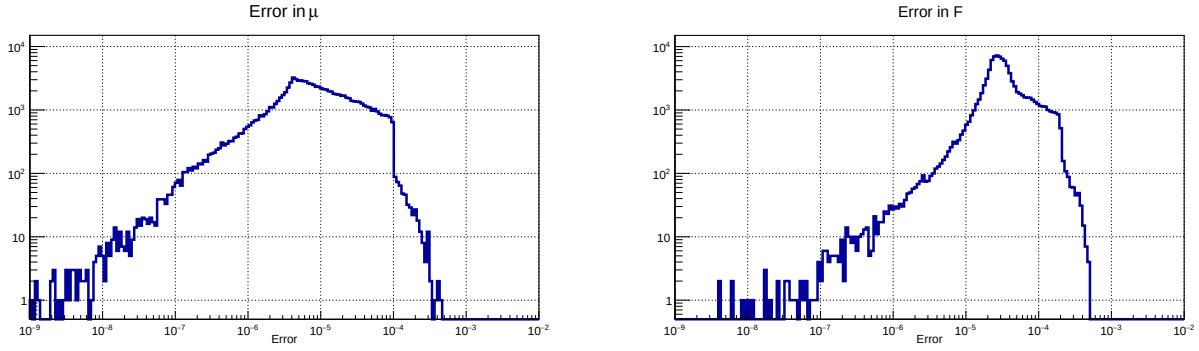

Figure 3: The error in μ and F from BuildCOM() of 10^5 points drawn randomly in $0 < \mu < 100$ and $0.1 < F < 1$, made with Example3().

should produce the plot shown in Fig. 3. The version of the look-up tables used to generate these plots is "Table4_Alpha". The function Example3() works in the following way:

1. First, the look-up tables are loaded by calling the function TableReader().
2. In a for-loop:
   (a) A random value of the mean mu_draw is chosen between 0 and 100, and a random value of the Fano factor F_draw is chosen between 0.1 and 1.
   (b) The function isSoft(mu, F, 0.001) is called to check if this pair of points falls on or below the Bernoulli modes to within 0.1%. Points below the Bernoulli modes are not possible, and points on the Bernoulli modes illicit the use of the Bernoulli distribution itself, so these points are skipped.
   (c) Then the line BuildCOM(mu_draw, F_draw, false); is used to define the distribution with the appropriate method. BuildCOM() will set the globally defined values of λ and ν as G_l and G_v (unless the method used is a Gaussian distribution or Bernoulli distribution).
   (d) Next, the mean and Fano factor obtained are calculated by calling Moments(G_l, G_v);. The resulting mean, variance, and Fano factor are stored globally as G_mu, G_var, and G_F respectively.
   (e) The relative error between the intended and obtained mean and Fano factor are calculated as:

$$\text{mu\_error} = \left|\frac{\text{mu\_draw} - \text{G\_mu}}{\text{mu\_draw}}\right| \times 100\% \qquad \text{and} \qquad \text{F\_error} = \left|\frac{\text{F\_draw} - \text{G\_F}}{\text{F\_draw}}\right| \times 100\%$$

3. The results from each iteration in the for-loop are stored and plotted as histograms like those shown in Fig. 3.

One can see that overall, the look-up tables and COM-Poisson code are accurate to 0.1% (and often much better) for all values of μ and F in this range.

4 Simulating the detection efficiency of an experiment

This example simulates the detection efficiency of an ionization-measuring device with the COM-Poisson code, similar to Fig. 4 of [1]. This involves drawing random numbers from a probability distribution defined for a given μ and F, and counting the proportion of random numbers that fall above a given threshold (set to $\mu = 4$ by default in this example), as a function of μ (here μ is both the mean number of ion pairs for a given distribution and also an energy scale with units of the number of pairs). The code to execute this example is contained in the function Example4(). Simply call this function in main(), re-compile Examples.cc, and run Examples.exe, which should produce the plot shown in Fig. 4. The function Example4() does the following:

1. First, the look-up tables are loaded by calling the function TableReader().
2. Then, in a for-loop for 1000 logarithmically spaced value of μ and a given value of F:
   (a) The line BuildCOM(mu, F, false); defines and builds the CDF of the probability distribution.

(b) 10^4 random numbers are drawn from this distribution using cdfDraw().

(c) These numbers are then re-drawn from a Gaussian distribution with $\sigma = 0.25$.

(d) The proportion of counts above the threshold of $\mu = 4$ is calculated and recorded.

3. All of the above is done in a for-loop to perform this for multiple values of the Fano factor ($F = 0.2, 0.4, 0.6, 0.8$, and 1).

4. The proportion of events above threshold (the "detection efficiency") is then plotted as a function of μ for each value of F, as in Fig. 4.

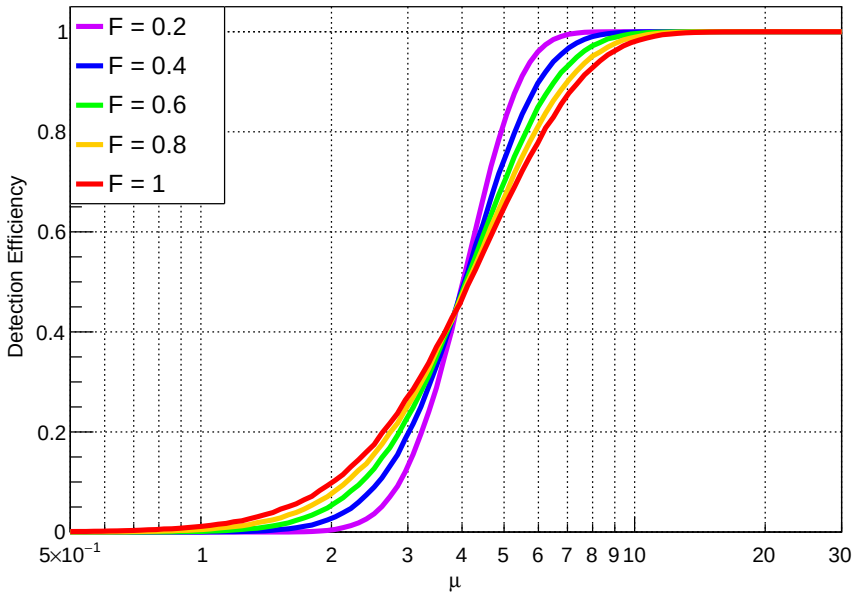

Figure 4: The simulated detection efficiency of an ionization measuring experiment with a threshold of $\mu = 4$ and Gaussian resolution with $\sigma = 0.25$, for different values of F, made with Example4().

---

[1] D. Durnford, Q. Arnaud, & G. Gerbier. "A novel approach to assess the impact of the Fano factor on the sensitivity of low-mass dark matter experiments". Accepted for Publication in Phys. Rev. D (2018). arXiv:1808.06967.