Counting statistics
The Significance of Errors

339 - Measurements Lab
Due February 5, 2018

January 23, 2018

Professor: David Cooke[1]
Teaching assistants: Aaron Mascaro[2], Andreas Spielhofer[3], Eli Martel[4], Ata Madanchi[5], Ziggy Pleunis[6]
Technician: Saverio Biunno

1 Introduction - PLEASE READ CAREFULLY

The purpose of this experiment is to demonstrate how the quality of data affects the significance of the conclusions that can be drawn from it. The analysis draws heavily on the ideas of basic statistics, so a review is strongly advised before starting the experiment. The experiment will produce a vast amount of information, therefore the manner in which you present this information is extremely important if you are to avoid swamping the reader. The problem of presentation is an integral part of any experiment.

The experimental measurement and the analysis procedure have been specifically chosen because the uncertainties associated with the measurements are purely statistical in origin and are readily determined. There is essentially no room for subjectivity in setting the error bars and therefore a rigorous statistical treatment is possible.

The data acquisition programs have been written for you. /mnt/resources/339/geiger contains all required components, copy it to your group directory by running (replace 'nn' with your station number):

```
cp -rf /mnt/resources/339/geiger /mnt/home/phys339-nn
```

Alternatively, they can be downloaded from my courses in a compressed .tar file geiger_2018.tar, which can be extracted to your group folder.

Now open the Arduino interface and load the Geiger_Counting.ino sketch. Upload it to the Arduino using the Sketch menu, Ctrl-U or the right arrow icon. You don't need to understand this code, but for the curious: the code says hello, then runs a loop that waits for a trigger pulse and restarts itself everytime the selected time interval elapses. Upon receiving this trigger, an interrupt routine runs that increments the number of clicks during that specific time interval. Afer each interval, the integer number of clicks is printed to the computer. This continues for the selected number of intervals.

The automation code is geiger.py. It is quite similar to the Introduction code in the way it interfaces with the Arduino. It is in the form used to collect the sample data set, sample.data.txt (yours will be saved as '$dateTimeStamp$'$\_GEIGER\_DATA.txt$ where 'datetimestamp' is in the form $YYYY\_MM\_DD\_HH\_mm\_SS$),

[1]david.cooke2@mcgill.ca
[2]aaron.mascaro@mail.mcgill.ca
[3]andreas.spielhofer@mail.mcgill.ca
[4]eli.martel@mail.mcgill.ca
[5]ata.madanchi@mail.mcgill.ca
[6]ziggy.pleunis@mcgill.ca

which is characterized in the Appendix A. Your programs should produce the same results when run using this dataset.

NOTE: we have included all of the insignificant digits from the analysis so that you can make an exact comparison with the output of your code. Be more selective when presenting your own final data.

The next piece of code you will run (for the second part of the experiment) is Geiger_Dwell_Time.ino. This code will measure the time between counts recorded by the Geiger counter (the dwell time). Again, understanding this code is not necessary, but here's a brief overview: the code sends a Hello message to the computer, then waits for an integer which will be interpreted as the number of milliseconds per interval. Upon receiving this integer, an interrupt routine is triggered everytime a pulse on pin 2 occurs, this causes a bin in a circular queue to be incremented. Once a bin is no longer active the content is printed back to the computer. There is a flag, debug. which will display some of the internal operations depending on the integer value assigned to it (1,2 or 3). For a detailed description on how the code works and how the data is acquired, see the geiger-software document, this may be of use especially if any problems arise (hopefully not the case!).

You will also notice that there are two python scripts (load_geiger_counting.py and load_geiger_dwell_time.py) that, when run, will load the data from the .txt files and plot figures with error bars. Running them as-is will grab some sample data as a demonstration.

2 Experimental Objectives

When a γ-ray photon strikes a Geiger counter, the counter responds by producing a 'click' sound. Switch on the counter provided and bring a radioactive source close to it. It is immediately clear that the source does not provide a steady stream of photons, they arrive in bunches of various sizes with gaps of variable length between them. Radioactive decay is probably the best example of a truly random process. In PHYS 257/258, you went through a similar Geiger counting experiment. This time you will analyze radioactive decay in a slightly different way: by first measuring the number of events in a fixed time interval, and then by measuring the dwell times between recorded events.

Each radio-isotope nucleus has some small probability, λ, of decay per unit time. Thus the total number of atoms decaying per unit time will be the product of this probability with the total number of atoms, since each decay reduces the number of original atoms:

$$\frac{d}{dt}N = -\lambda N \tag{1}$$

So:

$$N = N_0 e^{-\lambda t} \tag{2}$$

The half-life is defined as the time requried for 50% of the original atoms to have decayed, so

$$e^{-\lambda t_{1/2}} = \frac{1}{2} \tag{3}$$

Since the half-life is a well quantified value for known unstable nuclear isotopes, we can use this to determine the probability, λ, of decay per atom per unit time:

$$\lambda = -\frac{\ln \frac{1}{2}}{t_{1/2}} \tag{4}$$

PROBLEM 1: how many photons strike the Geiger counter in a particular time interval - say, one second for example? If you repeat the observation you will almost certainly obtain a different answer. Despite this variability, the average is a well defined quantity, and so is the frequency of the observations. For a given experimental arrangement you will obtain the same average count rate and distribution of counts.

PROBLEM 2: after one photon strikes the Geiger counter, how long will you have to wait before another arrives? If you repeat the observation you will be very likely (again) to obtain a different answer. Despite this, the average dwell time (or time between events) will be a well defined quantity. By measuring a large number of events we can build up a probability distribution that will allow us to extract the mean value.

WARM-UP EXERCISE: In this experiment we will be using a $^{137}_{55}Cs$ source with an activity of $5\mu Ci$ (where one Ci, or curie, is defined as 3.7×10^{10} decays per second, this is for the entire sample). Given this, make an estimate of the number of counts you would expect your Geiger counter to record in an interval of 0.2 seconds if it's placed 5 cm from the source. Also make an estimate of the average dwell time (time between events) that your Geiger counter will record for the same distance. You will have to make some approximations, so an order of magnitude estimate is appropriate.

3 Counting Experiment

You should have the Gieger_Counting.ino sketch loaded to the Arduino. Now place the source 5-10 cm away from the Geiger counter and connect its BNC output to the PIN 2 input on the panel. In geiger_counting.py select one run with a time of 1 second for 50-100 intervals. The program counts the 'clicks' for 1 second then increments the appropriate bin in memory. The process is repeated for the selected number of intervals then the program stops. The display shows how many times a given number of counts occured in the selected interval. Try calculating the mean and variance of the data.

There is a python script (load_gieger_counting.py) that has the basics for loading and plotting the histogram with rudimentary errorbars (the standard deviation), this should help get you started.

What can you say about the shape of the distribution? Try comparing your data to standard forms (e.g. Gaussian or Poisson) by plotting your data with the corresponding calculated form on top of it. Can you say which is a better representation of the data? What other distributions are possible? Are they any better? What distribution did you expect? Why?

4 Dwell Time Experiment

Now it's time to load the Geiger_Dwell_Time.ino sketch to the Arduino. In geiger_dwell_time.py, set the number of events to be recorded (N) to 100 and the number of replicas to record to 1 (this should be around line 100 in the code) and run it. The program then counts the time between 'clicks', displays these in a histogram, and saves them to an array for later analysis. The process is repeated for the selected number of events, then the program stops.

What can you say about the shape of the distribution? Try comparing your data to standard forms (e.g. Gaussian or Poisson) by plotting your data with the corresponding calculated form on top of it. Do either of these make sense? (Hint - they shouldn't!)

The data you've collected should look like a fairly familiar curve. Remember that for a process described by a Poisson distribution, the most likely time of occurrence for an event is immediately following a prior event. If the arrival of γ rays at your Geiger counter from radioactive decay is a Poisson process, then why doesn't the Poisson distribution fit the data you've collected? That's because in this case, you've measured the time *between* events and not the number of events *per unit time* as you did for the counting experiment. What distribution *should* you be using, and what information can you extract from it?

5 Data Analysis

In order to show that the measured data follow one form better than some others it is necessary to have a more objective test of similarity. This test must answer these three questions:

- how similar is the data set to the expected distribution?
- how significant is this similarity given the uncertainty in the data?
- how likely is it that you would obtain the data like yours if the underlying distribution had the assumed form?

Many statistical tests exist with characteristics that make them more or less attractive / suitable for a specific problem and you are free to use as many as you wish to obtain your results. One standard statistical method is the or chi-squared test which compares the difference between the observed and expected values at each point, to the uncertainty in the measurement of that point. For data in the form of a histogram a common definition of the reduced for a data set is given by:

$$\chi^2 = \frac{1}{n}\sum_{i=1}^{n}\frac{(O_i - E_i)^2}{E_i} \tag{5}$$

where the sum runs over all the n bins in the histogram. O_i and E_i are the observed and expected values respectively in bin i.

A more general form that makes fewer assumptions about the statistical properties of the data may be used for non-histogram data:

$$\chi^2 = \frac{1}{n}\sum_{i=1}^{n}\frac{(O_i - E_i)^2}{\sigma_i^2} \tag{6}$$

where σ_i^2 is the variance of the observed value.

O_i and E_i are readily available, they are your data and the distribution you want to compare it with. σ_i^2 is more difficult – how to determine the error on the contents of a bin in your histogram? To solve the problem we turn to an invaluable technique for establishing errors or reliability – replication.

The χ^2 has a probability distribution which depends only upon the number of degrees of freedom, ν. In Python this distribution is available as scipy.stats.chi2.pdf(χ^2,ν). Given a huge data set we could compare the distribution of χ^2 with the expected distribution directly. With a smaller dataset, a frequently used technique is to select a point on the distribution and verify that approximately the correct fraction of χ^2 values fall on either side of this point. In Python there is a function scipy.stats.chi2.isf(α,ν) which will return the value β where

$$\int_{\beta}^{\infty} P_\nu(\chi^2)d\chi^2 = \alpha \tag{7}$$

That is the value, β, which the probability of a χ^2 being great than β, for a given number of degrees of freedom, ν, is α.

If you repeat the same experiment a large number of times, then, if the replicas are truly independent, the scatter in your results should reflect the uncertainties in the experiment. Furthermore, if you obtain a larger scatter than you expect from a naive estimation of the uncertainties in your measurements, it is highly likely that either your estimates are wrong, or that there are additional sources of error in the procedure. In principle, if the analysis is done correctly, and all sources of error are included, the two procedures will give the same result. However there is no way of being sure. Whenever possible, replication, rather than guestimates, should be used to establish limits of precision.

6 Gathering Statistically Useful Data

Place the source so that you get an average of 7 counts in 0.2 seconds (once placed, do NOT move the source until you've completed this and the following section). Reload the Arduino with the **Geiger_Counting.ino** sketch, then use **geiger_counting.py** to count for 64 intervals of 0.2 seconds, and repeat this measurement 128 times. It will create a 2-d array with one replica per row in histogram form. Since the replicas are equivalent and

independent, you can determine the mean and variance for each bin from the data set. Use the set of variances to perform a χ^2 test on each replica by comparing each one in turn with Gaussian and Poisson distributions. The mean and variance for the test distributions may be calculated from the sets of means that you have obtained above for each bin. Can you decide between the two forms for your data?

The test fails because your data is too noisy - the uncertainties are to large. You may improve the data in two equivalent ways:

- collect more intervals than the 64 recommended
- add several replicas together.

The latter is preferred since you will see the improvement in quality within the original data set.

It should be apparent that if the first 2 rows of the data set are added together columnwise, then the resulting single row would be identical to the replica which would have resulted if the acquisition had been performed with twice the number of intervals per replica. If this is repeated through out the dataset the resulting dataset will have half the number of replicas of the original, but the total number of intervals contained will be unchanged. For computational purposes with Python it is not necessary to extract alternating replicas, since they are independent, it would suffice to add the first half to the second half.

```
L = numpy.size(original,axis=0)/2
new_array = original[:L,:]  + original[L:]
```

Repeat the analysis on the compressed dataset. Can you now distinguish the two distributions? If you plot out one replica as a histogram you should see that is much smoother with smaller variances.

Repeat the collapse of the data set and compare again.

How good does the data set have to be before you can tell the difference between Gaussian and Poisson distributions?

Finally, take means for each column determined in the first step and perform a χ^2 test on this set. What should you use as the variances for this data set? (It is not the variance you calculated with this set of means.)

You have shown that the output from a radioactive source follows a Poisson distribution. What can you say about the average number of γ-rays arriving at your detector in 0.2 seconds? What is the uncertainty on this estimate?

7 Counting vs. Dwell Time

To compare the two measurement techniques we'll go back again to the dwell time measurement by loading the Arduino with the **Geiger_Dwell_Time.ino** sketch and running **geiger_dwell_time.py**. This time, set the number of events to 450 and replicas to 128. This should result in gathering approximately the same amount of data as your previous measurement using the Counting scripts.

Once you've acquired this data, repeat the analysis above by fitting to the appropriate distribution (hint: see Equation 2). How good does the data have to be before you can demonstrate that the distribution is exponential? What can you say about the mean dwell time (λ), and what is it's uncertainty?

Now we're going to test two more 'regimes' of these measurement techniques: the **high count-rate** regime, and the **low count-rate** regime.

First start with the high count-rate data by placing the source about 3cm away from Geiger counter and running both the Counting experiment and the Dwell Time experiment with the same parameters above. The

mean of the counting data should be quite high (∼20 for 0.2s intervals). How well can you determine the mean by fitting a Poisson distribution to this data, i.e. is your uncertainty better or worse, and what happens to your χ^2 value? How about your Dwell Time data, does your exponential distribution fit better or worse, and what happens to the uncertainty on the mean dwell time?

Now run both experiments again with a much lower count rate of around 3-5 counts per 0.2s interval. What happens to the χ^2 value and the uncertainty on the mean for your Counting data? Does the exponential distribution fit better or worse for the dwell time data, and what happens to the uncertainty on *that* mean value?

If you noticed a difference between these measurements in the low/high count regimes, discuss where they arise from.

8 Definitions of Statistical Parameters

All of the statistical calculations in your program should be done using double precision reals (double). The accumulated errors from working in single precision (float) can make your results meaningless and the extra computer time required is very small.

Mean:

$$< x >= \frac{1}{n}\sum_{i=1}^{n} x_i \tag{8}$$

Variance:

$$\sigma^2 = \frac{1}{n}\left[\sum_{i=1}^{n} x_i^2\right] - < x >^2 \tag{9}$$

This is equivalent to the usual definition:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}\left[x_i - < x >\right]^2 \tag{10}$$

but it is more efficient for computer calculation since it only requires a single pass through the data. Some people object to the approximation used in the first form, but for the data you will working with the effects should be negligible. If you are in any doubt, try both.

The standard deviation is simply σ.

The standard error is generally, but not necessarily, two standard deviations. (Why?) The form of error used and the method used to estimate it should always be specified when reporting results.

The probability of getting ν counts in a time interval for a process following a Poisson distribution is given by:

$$P_\mu(\nu) = \frac{\mu^\nu}{\nu!}e^{-\mu} \tag{11}$$

where ν is the bin number and μ is a positive parameter. For this distribution, the mean and variance are the same and equal μ. One parameter specifies the full distribution form.

Similarly, if the process follows a Gaussian distribution:

$$P_{\mu,\sigma}(\nu) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left[\frac{\nu-\mu}{\sigma}\right]^2} \tag{12}$$

where μ is the mean and σ^2 is the variance of the distribution.

NOTE:

1. Both of these distributions are normalized to unit area. To compare them with your data you will have to multiply by the number of points in your data set. For data with a small mean, comparison with the Gaussian will require a different normalization. A Python function which may be of utility in this case is *scipy.stats.norm.sf(*β, μ, σ*)* which gives the value α, where

$$\alpha = \int_{\beta}^{\infty} P_{\mu,\sigma}(\nu) d\nu \tag{13}$$

9 Big Picture Overview

The following is an overview of what the χ^2 test looks like in action. I would strongly suggest that you collect data sets with the number of replicas being a power of 2. This way you can compress them and choose a fraction which will map to an integer number of replicas. A given data set will have a fixed number of degrees of freedom, it does not matter how it is compressed. In this case I have chosen 1 as 8 the fraction of the total number of replicas for which the ?2 would be greater than the test value, if all the assumptions about the data and the distributon are correct. The test values of the χ^2 are calculated. The χ^2 values calculated for each replica and distribution are compared against the test values. If the actual fraction of χ^2 values differs significantly from the expected value, then it follows one or more of the assumptions made is false. In the example shown, since the Poisson fraction is not significantly different from the expected 1, but the Gaussian fraction differs greatly, it would be hard to contradict 8 my claim that the probability distribution of events per interval is not a Gaussian distribution.

```
   Dataset has 20 degrees of freedom
12.5% of Poisson chisq should be larger than X_pc = 26.189306
12.5% of Gaussian chisq should be larger than X_gc = 24.996958
...
512 replicas with 64 intervals each:  16% of X_p > x_pc, 20% of X_g > X_gc
256 replicas with 128 intervals each:  14% of X_p > x_pc, 26% of X_g > X_gc
128 replicas with 256 intervals each:  18% of X_p > x_pc, 44% of X_g > X_gc
64 replicas with 512 intervals each:  9% of X_p > x_pc, 75% of X_g > X_gc
32 replicas with 1024 intervals each:  12% of X_p > x_pc, 97% of X_g > X_gc
16 replicas with 2048 intervals each:  12% of X_p > x_pc, 100% of X_g > X_gc
```

The relevant code is shown below. This is merely given as an overview of how you might proceed. I do strongly recommend you implement the scatter plot of column statistics, it provides an answer to a question. 'cd' is compressed data (initially '*new_big*'), which could be the array passed back from the data acquisition code. In this case it is actually that data with the zero filled columns removed, but that's not relevant to the code shown, so long as the 'events' vector is similarly trimmed.

```
fraction = 1./8.
chisq_cut_p = scipy.stats.chi2.isf(fraction,dof-1)
chisq_cut_g= scipy.stats.chi2.isf(fraction,dof-2)
print("Dataset has %d degrees of freedom"%(dof))
print("%.1f%% of Poisson chisq should be larger than X_pc = %f"%(100*fraction,chisq_cut_p))
print("%.1f%% of Gaussian chisq should be larger than X_gc = %f"%(100*fraction,chisq_cut_g))
p.loglog()  # for the scatter plot of column stats
p.xlabel("column means")
p.ylabel("column variances")
cd = new_big
while fraction*numpy.size(cd,axis=0) > 1:
    rows = numpy.size(cd,axis=0)
    cols = numpy.size(cd,axis=1)
    intervals = cd.sum()/rows
    row_means,row_vars = calc_row_stats(events,cd)
    col_means,col_vars = calc_col_stats(cd)
    p.scatter(col_means,col_vars)
    p.pause(1e-3)
    f_poisson, f_gaussian = gen_distributions(events,intervals,row_means, row_vars)
    chisq_p = calc_chisq(cd,f_poisson,col_vars)
    chisq_g = calc_chisq(cd,f_gaussian,col_vars)
    fraction_p = (chisq_p > chisq_cut_p).mean()
    fraction_g = (chisq_g > chisq_cut_g).mean()
    print("%d replicas with %d intervals each: %.0f%% of X_p > x_pc, %.0f%% of X_g > X_gc"%
    (rows,intervals,100*fraction_p,100*fraction_g))
    save_intervals.append(intervals)
    save_fraction_ps.append(fraction_p)
    save_fraction_gs.append(fraction_g)
    cd = crunch(cd)
```

Figure 1: Python code to analyze

10 Suggested References

1. John R. Taylor, “An Introduction to Error Analysis”
2. Phillip R. Bevington, “Data Reduction and Error Analysis for the Physical Sciences”
3. William H. Press et al “Numerical Recipes” (Mainly Chapter 13)

This last book is an extremely useful general reference work for scientific computing, and well worth adding to your collection.

Replica Statistics			Column Statistics		
Replica	Mean	Variance	Column	Mean	Variance
0	3.81000000000	5.07390000000	0	6.10000000000	6.89000000000
1	3.23000000000	3.47710000000	1	12.80000000000	11.56000000000
2	3.90000000000	4.07000000000	2	17.65000000000	15.42750000000
3	3.18000000000	4.84760000000	3	18.55000000000	8.74750000000
4	3.36000000000	3.45040000000	4	16.60000000000	13.74000000000
5	3.49000000000	4.34990000000	5	13.20000000000	7.16000000000
6	3.28000000000	5.52160000000	6	6.75000000000	6.58750000000
7	3.36000000000	3.51040000000	7	4.25000000000	4.48750000000
8	3.40000000000	3.90000000000	8	2.45000000000	2.04750000000
9	3.03000000000	4.50910000000	9	0.95000000000	0.74750000000
10	3.27000000000	3.93710000000	10	0.35000000000	0.32750000000
11	3.41000000000	4.30190000000	11	0.15000000000	0.12750000000
12	3.22000000000	4.11160000000	12	0.15000000000	0.22750000000
13	3.82000000000	4.14760000000	13	0.00000000000	0.00000000000
14	3.83000000000	4.80110000000	14	0.05000000000	0.04750000000
15	3.58000000000	5.00360000000	15	0.00000000000	0.00000000000
16	3.23000000000	3.63710000000	16	0.00000000000	0.00000000000
17	3.27000000000	4.83710000000	17	0.00000000000	0.00000000000
18	3.34000000000	5.98440000000	18	0.00000000000	0.00000000000
19	3.43000000000	4.28510000000	19	0.00000000000	0.00000000000

data.pdf

Figure 2: Characteristics of sample.data

A Characteristics of sample.data

B Resources

The data acquisition programs geiger_counting and geiger_dwell_time are provided to you. The manual pages for the program and for its output format are attached as an unlabeled appendix to this handout.

You are provided