Statistics of Radioactive Decay

Last modified by E. Eyler, October 27, 2005.
Based on earlier labs by E. Eyler and D. Hamilton.

The statistical behavior and probability distribution for counting random radioactive decay events is measured using a Geiger counter with computer-assisted data acquisition. First, counts are accumulated for fixed time periods, to study the population statistics. Then the time distribution of the counts is studied in detail, so that histograms of the counting frequency distribution can be prepared for comparison to the Poisson and Gaussian probability distribution functions.

I. INTRODUCTION

I.A. RADIOACTIVE DECAY

The atom is composed of a central core, or nucleus, and various electrons in orbit about it. Nuclear transitions from higher to lower energy states are accompanied by the emission of either electromagnetic radiation or subatomic particles. This phenomenon is called “radioactivity” and is exhibited by naturally occurring elements (e.g., uranium and radium) as well as man-made isotopes.

Radioactive substances undergo three common types of decay: alpha (α) decay, where the emitted particles are ^{4}He nuclei, beta (β) decay in which the emitted particles are either electrons or positrons, and gamma (γ) decay, where the emitted radiation is a high-energy photon. In alpha decay the parent nucleus loses four units of mass and two units of charge, and thus the resulting daughter nucleus is that of a different element. When a nucleus undergoes beta decay, the daughter nucleus has the same number of nucleons (mass number) as the parent but the charge number is changed by one. Emitted simultaneously with the beta particle is a neutral particle with a near zero rest mass called the neutrino. The parent nucleus is often left in an energetically excited state following an alpha or beta decay process. Transitions from higher to lower nuclear energy states of the same isotope are accompanied by the emission of gamma radiation. These gamma “rays” are identical in nature to x rays, visible light, radio waves and other forms of electromagnetic radiation, except that the energies of gamma ray photons are much higher and their wavelengths are much shorter.

A radioactive event is physically evident to us only in the extent to which the decay products interact with a detector to produce experimentally measurable effects. The principal effect is that of the ionization of atoms along the path of the emitted particle. The ionization is due to the energy transferred to the atom in the collision with the particle emitted in the radioactive event. Most detectors of radioactivity utilize this ionization process to register the occurrence of a radioactive event.

I.B. OPERATING PRINCIPLE OF A GEIGER-MÜLLER TUBE

A Geiger-Müller (GM) tube is used in this lab to detect beta particles and gamma rays; alpha particles cannot penetrate the metal window of the tube. The construction of the tube is illustrated in Fig. l. The two electrodes of the GM tube are connected to a high voltage power supply. The voltage produces a radial electric field inside the tube. When a β particle or a γ ray passes through the tube, it can ionize one or more of the atoms inside the tube and thereby produces negatively charged electrons and positively charged ions. The electrons are then accelerated by the electric field toward the positively charged anode. These rapidly moving electrons then collide with other gas atoms producing more and more free electrons and ions. The result is an electron avalanche, and an intense pulse of electrons reaches the center anode wire in response to a single ionization event. This pulse is easily detected and counted by suitable electronics. You might wonder why the discharge does not continue, turning into a steady arc rather than a pulse. The solution is that the GM tube discharge is designed to self-quench, by addition of electronegative gases that eventually absorb excess free electrons.

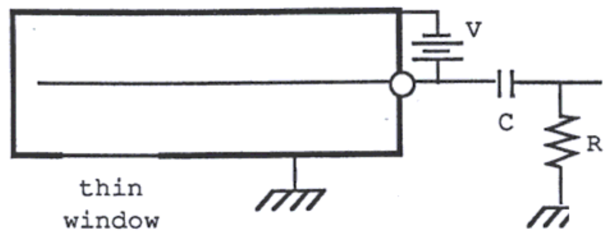

Figure 1. The positive center terminal of the GM tube attracts electrons resulting from the discharge avalanche.

I.C. THE POISSON PROBABILITY DISTRIBUTON

The basis for the statistical treatment of the data to be obtained in this experiment is the Poisson probability distribution, which gives the probability of observing a given number of counts per time interval. The Poisson distribution applies only if the counts occur in a random manner and the average count rate R remains constant during the measurement. The probability of measuring n counts during a time interval T is

$$P(n)=\frac{(RT)^{n}e^{-RT}}{n!} \tag{1}$$

The probability function is normalized so that

$$\sum_{n=0}^{\infty} P(n) = 1 \tag{2}$$

In general, the mean value of *n* is defined as the first moment of *P*(*n*),

$$\langle n \rangle = \sum_{n=0}^{\infty} n\, P(n) \tag{3}$$

and the standard deviation is the square root of the second moment about the mean,

$$\sigma = \sqrt{\sum_{n=0}^{\infty} \left(n - \langle n \rangle\right)^2 P(n)} \tag{4}$$

For the case of the Poisson distribution the results are particularly simple:

$$\sigma^2 = \langle n \rangle = RT \tag{5}$$

Note that the result for the mean is obvious, because we stipulated above that the average count rate is a constant value *R*.

For large value of *n*, the Poisson distribution of Eq. (1) is closely approximated by the Gaussian probability distribution. This can be demonstrated by using Sterling's approximation,

$$n! \simeq n^n\, e^{-n} \sqrt{2\pi n}\,, \tag{6}$$

to prove that for large *n*,

$$P(n) \simeq G(n) = \frac{\exp\left[\dfrac{-\left(n - \langle n \rangle\right)^2}{2\langle n \rangle}\right]}{\sqrt{2\pi n}}\,. \tag{7}$$

Optional question, for extra credit: Show this. *Hint*: This is not so easy. You may wish to take ln(*P(n)*) and use the approximation $\ln(1+x) \simeq x - x^2/2$).

II. MEAN AND STANDARD DEVIATION OF THE COUNT RATE

To get started, develop a feel for the behavior of a G-M tube by using one of the old Sargent-Welch counters, if one is available. If so, use several different detector voltages while counting for fixed time intervals with the aid of a stopwatch (a computer interface is available as an alternative, but it is not necessary). Can you find the “plateau” region in which the G-M tube has a count rate nearly independent of voltage? What minimum voltage is needed to assure that no more than 10-20% of the counts are missed? If you have both a β and a γ source available, note the dramatic difference between them in their response to a thin layer of shielding material.

Next, use either type of Geiger counter, preferable the newer Medcom unit, in conjunction with the program `ctr_evt.exe`. If you know from previous experiments the dead time of your detector, the time during which it is nonresponsive after each count, enter this information on the program control window---otherwise enter zero. Set the "Number of repetitions" to twenty, so that the program will acquire counts during twenty time intervals, each of duration *T*. It will then display histogram data, which you should ignore for now, along with the mean count,

$$\langle n \rangle = \frac{1}{20}\sum_{i=1}^{20} n_i = \frac{T}{20}\sum_{i=1}^{20} R_i, \tag{8}$$

and the standard deviation of the distribution as estimated from the 20 measurements,

$$\sigma \simeq s_{n_i} = \sqrt{\frac{1}{19}\sum_{i=1}^{20}\left(\langle n \rangle - n_i\right)^2}. \tag{9}$$

You should record the results for counting intervals *T* of 1, 2.5, 5, 10, and 25. Make a log-log plot of the value σ as a function of *T* and fit the data to a straight line (In practice, you will probably want to perform a linear regression on the log of σ vs. the log of *T*). Compare σ to the value predicted from the rate, $\sqrt{\langle R \rangle T}\,/\,T$. Discuss the results of the measurement. Is σ the uncertainty in the mean count rate or in the individual rates R_i? Use your result for σ to derive the other of this pair of uncertainties.

III. MEASURING THE COUNT FREQUENCY DISTRIBUTION

III.A. EXPERIMENTAL PROCEDURE

If using the older Geiger counters, set the high voltage for the GM tube to a point in the middle of the plateau. The newer Medcon counters do not require adjustment. Place a radioactive source under the GM tube, opening the window if it has one. The output should be connected to the interface circuitry for the computer, which is slightly different for the two counter types. Run the program `ctr_evt.exe` to determine the count rate. The program, which was written in the LabWindows environment from National Instruments, uses counters on the data acquisition boards both to collect the data and to control the timing. A copy of the key portions of this program is attached for reference purposes, but you will not need to read or modify it unless you wish to extend or change the experiment in some way. If you know it, enter the dead time, which will be used as a (small) correction by the program. After starting the program, move the sample until the counting rate is about 10 to 20 counts per second. If you know from previous experiments the dead time of your detector, the time during which it is nonresponsive after each count, enter this information on the program control window---otherwise enter zero.

Now use the program to acquire data that you will use to analyze the probability distribution of the radioactive decay events. The program displays and records a “frequency distribution” histogram, a bar graph that records the number of instances for which n events are counted during a series of fixed counting time intervals, each of duration T. You can set T, as well as the number of counting intervals to collect before the program stops, using the program control window. Experiment with the parameters for a few minutes until you both understand the behavior of the data acquisition program, and obtain a visually pleasing result.

The program allows you to store on disk both the raw counting data (the number of counts observed during each interval) and the histogram (in the form of a list of frequencies, and information on the “bins” that specify how many count values n were lumped together in each bar of the histogram). The program also displays, but does not record, the time interval, the mean count rate, and the standard deviation---*be sure to record these numbers for further use*. In your analysis, you should work with the raw data, using Mathcad or Matlab to create a new histogram better suited to your specific requirements. Mathcad has functions called “hist” and “histogram” to assist you in this task. The help menu for these functions also references a Quicksheet showing an example of their use. Matlab has both a versatile histogram plotting function, “hist,” and a histogram analysis function for matrices of data, “histc.”

For your first data set, collect enough data to construct a good histogram with a value for the time window T that is roughly $2/R$, where R is the average count rate.

Now run the program a second time, but with a much higher value for the mean number of events, RT. To do this, enter a value for T that is about $15/R$. Then run the program a third time with T=$5/R$. When you are finished, turn off the voltage to the GM tube, or for the Medcom detectors, turn them off completely.

III.B. DATA ANALYSIS

Create a histogram showing the frequency at which each count rate n occurs (i.e., make a histogram with a bin size of one). Now normalize the area of the of the histogram: if $Y(n)$ is the height of the n’th bar, meaning that n counts were observed during an interval of duration T on Y different occasions, you should multiply all of your results by a constant C, chosen so that

$$C\sum_{n=0}^{\infty} Y(n) = 1 \tag{10}$$

In practice, the sum extends only to a finite value of n, because there will be some value $n_{highest}$ that is the largest number of counts that you ever observed during any time interval T, so for all higher values of n, Y=0.

The normalized histogram can be interpreted directly as an estimate of the normalized probability distribution $P(n)$ for observing n events during time T. Include error bars with each data point (you may want to discuss with your TA how to go about estimating these). Calculate $\langle n \rangle$, the mean number of counts per time interval, and its uncertainty, from the total accumulated count, the total elapsed time, and the value of the time interval. Then calculate a value for $\langle n \rangle$ directly from Eq. (3).

Compare each set of data to the Poisson distribution function. For the large n data set, also compare your results to the Gaussian distribution. To facilitate this comparison, overlay graphs of these distribution functions on your data sets. Calculate the standard deviation σ from your value of $\langle n \rangle$ as well as from Eq. (4). Indicate $\langle n \rangle$ and σ on your graphs. Also calculate the mean count rate $\langle R \rangle$ in counts/sec from your results.

How well do the two distribution functions describe the data? Do the deviations between the Poisson function and your results correspond well with the size of your error bars?

```c
#include <formatio.h>
#include <analysis.h>
#include <Dataacq.h>
/**********************************************************************/
/*
/*      ctr_evt_E.c
/*
/*  This CVI LabWindows program uses an DAQ-STC counter on an E-series
/*  DAQ card to count TTL edges of the signal on the Counter 2 SOURCE pin.
/*
/*  It saves three output arrays:
/*     filename.txt: counter data, corrected for dead time.
/*     filename.hsx: Histogram x-axis (contains center channel #'s for each bin).
/*     filename.hst: Histogram data (corrected event counts for each channel).
/*
/*  The timing is currently done by the sytem clock.  This is not really
/*    optimal.  A better method would be to use another counter to gate counter
/*    2 with a precisely determined interval.  This enhancement was omitted
/*    for the moment due to development time limitations.
/*
/*    Last modified 10/27/05 by E. Eyler
/*                                                                     */
/**********************************************************************/


#include <cvirte.h>    /* Needed if linking in external compiler; harmless otherwise */
#include <ansi_c.h>
#include <userint.h>
#include <utility.h>
#include "easyio.h"
#include "ctr_evt.h"


/* For simplicity, make all the key variables global, even though it's not terrific style */

static int evtCount;
static int daqError = 0;
static unsigned long taskID = 0;
short device;
int counter;
static int first_time = 1;
long count;
long initial_count;
unsigned short repetitions;
unsigned short current_rep;
double interval, dead_time;
double *results;        // Allocated in StartCallback
int hist_intervals=0;
double *axisArray;      // Allocated in CreateHistogram
int *histogramArray;    // Allocated in CreateHistogram

/* We need a few forward declarations of utility functions: */

static int CtrDisplayError (short error);
void Stop_Counting(void);
int CreateHistogram(double *results,int current_rep);

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)    /* Needed if linking in external compiler; harmless otherwise */
        return -1;    /* out of memory */

    evtCount = LoadPanel (0, "ctr_evt.uir", EVT_COUNT);
    DisplayPanel (evtCount);
    SetCtrlAttribute (evtCount, EVT_COUNT_TIMER, ATTR_ENABLED, 0);
    SetCtrlAttribute (evtCount, EVT_COUNT_TIMER_DISPLAY, ATTR_ENABLED, 0);
    RunUserInterface ();

    return 0;
}


int CVICALLBACK StartCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
```

```

char strCounter[10];

    switch (event) {
        case EVENT_COMMIT:
            GetCtrlVal (evtCount, EVT_COUNT_DEVICE, &device);
            GetCtrlVal (evtCount, EVT_COUNT_COUNTER, &counter);
            GetCtrlVal (evtCount, EVT_COUNT_INTERVAL, &interval);
            GetCtrlVal (evtCount, EVT_COUNT_REPS, &repetitions);
            GetCtrlVal (evtCount, EVT_COUNT_DEAD_TIME, &dead_time);
            sprintf (strCounter, "%d", counter);

            current_rep = 0;

            if (!first_time) free (results);
            first_time = 0;
            results = malloc(repetitions*sizeof(double));

            daqError = CounterEventOrTimeConfig (device, strCounter, ONE_COUNTER,
                                                 USE_COUNTER_SOURCE,
                                                 COUNT_CONTINUOUSLY,
                                                 COUNT_ON_RISING_EDGE,
                                                 UNGATED_SOFTWARE_START, &taskID);
            if (CtrDisplayError (daqError))
                return daqError;

            /* Start the timer */

            initial_count = 0;
            daqError = CounterStart (taskID);

            SetCtrlAttribute (evtCount, EVT_COUNT_TIMER, ATTR_INTERVAL, interval);
            SetCtrlAttribute (evtCount, EVT_COUNT_TIMER, ATTR_ENABLED, 1);

            if (CtrDisplayError (daqError))
            return daqError;

            SetCtrlAttribute (evtCount, EVT_COUNT_TIMER_DISPLAY, ATTR_ENABLED, 1);

            SetCtrlVal (evtCount, EVT_COUNT_CURRENT_REP, current_rep);
            SetCtrlVal (evtCount, EVT_COUNT_READING, 0);

            SetInputMode (evtCount, EVT_COUNT_DEVICE, 0);
            SetInputMode (evtCount, EVT_COUNT_COUNTER, 0);
            SetInputMode (evtCount, EVT_COUNT_INTERVAL, 0);
            SetInputMode (evtCount, EVT_COUNT_REPS, 0);

            break;
    }
    return 0;
}


int CVICALLBACK StopCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{

    char *fname,*pathname;
    int status;

    switch (event) {
        case EVENT_COMMIT:

            Stop_Counting();

            fname = malloc(256*sizeof(char));
            pathname = malloc(256*sizeof(char));
            GetCtrlVal (evtCount, EVT_COUNT_FILENAME,pathname);
            strcpy(fname,pathname);
            strcat (fname, ".txt");
//            FileSelectPopup ("c:\\p258", "*.txt", "", "data", VAL_OK_BUTTON, 0, 0,
//                       1, 1, fname);


            /* Write the raw counter data to "filename.txt" */
```

```
            status = ArrayToFile (fname, results, VAL_DOUBLE, current_rep, 1, VAL_GROUPS_TOGETHER,
                         VAL_GROUPS_AS_COLUMNS, VAL_CONST_WIDTH, 10, VAL_ASCII,
                         VAL_TRUNCATE);
            if (status<0) printf("Error in writing file, status is %i\n",status);

          /* Now write the histogram files (array of center channels, array of histogram freq's) */

          if (hist_intervals != 0) {
            strcpy(fname,pathname);
            strcat (fname, ".hsx");
            status = ArrayToFile (fname, axisArray, VAL_DOUBLE, hist_intervals, 1,
                VAL_GROUPS_TOGETHER,VAL_GROUPS_AS_COLUMNS, VAL_CONST_WIDTH, 10, VAL_ASCII,
                 VAL_TRUNCATE);
            if (status<0) printf("Error in writing hist. x file, status is %i\n",status);

            strcpy(fname,pathname);
            strcat (fname, ".hst");
            status = ArrayToFile (fname, histogramArray, VAL_INTEGER, hist_intervals, 1,
                VAL_GROUPS_TOGETHER,VAL_GROUPS_AS_COLUMNS, VAL_CONST_WIDTH, 10, VAL_ASCII,
                VAL_TRUNCATE);
            if (status<0) printf("Error in writing histogram file, status is %i\n",status);
          }
          free(fname);
          free(pathname);

        break;
    }
    return 0;
}

int CVICALLBACK TimerCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    double rate, corrected_count, mean, std_dev;
    short overflow;
    long temp;


    switch (event) {
        case EVENT_TIMER_TICK:
            daqError = CounterRead (taskID, &count, &overflow);

            if (CtrDisplayError (daqError))
                return daqError;

            temp = count;
            count -= initial_count;

            /* Save the initial count to start a new interval */

            initial_count = temp;

            corrected_count = count/(1-count*dead_time/interval);
            results[current_rep++] = corrected_count;
            rate = corrected_count/interval;
            SetCtrlVal (evtCount, EVT_COUNT_READING, corrected_count); /* Display final results */
            SetCtrlVal (evtCount, EVT_COUNT_RATE, rate);
            SetCtrlVal (evtCount, EVT_COUNT_CURRENT_REP, current_rep);
            StdDev (results, current_rep, &mean, &std_dev);
            SetCtrlVal (evtCount, EVT_COUNT_MEAN, mean);
            SetCtrlVal (evtCount, EVT_COUNT_STD_DEV, std_dev);

            hist_intervals = CreateHistogram(results,current_rep);

            if (current_rep==repetitions)
              Stop_Counting();

            break;
    }
    return 0;
}


int CVICALLBACK Timer_DisplayCallback (int panel, int control, int event,
```

```
    void *callbackData, int eventData1, int eventData2)
{
  short overflow;

  switch (event)
    {
    case EVENT_TIMER_TICK:
      daqError = CounterRead (taskID, &count, &overflow);
      if (CtrDisplayError (daqError))
        return daqError;

      count -= initial_count;

      SetCtrlVal (evtCount, EVT_COUNT_READING, (double)count);
      break;
    }
  return 0;
}

int CVICALLBACK QuitCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:

            Stop_Counting();

            QuitUserInterface (0);
            break;
    }
    return 0;
}


static int CtrDisplayError (short error)
{
    static int lastError = 0;


    if (error != lastError)
    {

        if (error == 0)
        {
            SetCtrlAttribute (evtCount, EVT_COUNT_ERRORBOX, ATTR_TEXT_COLOR,
                              VAL_BLACK);
        }
        else
        {
            SetCtrlAttribute (evtCount, EVT_COUNT_ERRORBOX, ATTR_TEXT_COLOR,
                              VAL_RED);

        }
        ResetTextBox (evtCount, EVT_COUNT_ERRORBOX,
                        GetDAQErrorString(error));
    }
    lastError = error;
    return error;
}

void Stop_Counting(void)
{


        daqError = CounterStop (taskID);
        if (CtrDisplayError (daqError))
          return;


        SetCtrlAttribute (evtCount, EVT_COUNT_TIMER, ATTR_ENABLED, 0);
        SetCtrlAttribute (evtCount, EVT_COUNT_TIMER_DISPLAY, ATTR_ENABLED, 0);

        SetInputMode (evtCount, EVT_COUNT_DEVICE, 1);
        SetInputMode (evtCount, EVT_COUNT_COUNTER, 1);
        SetInputMode (evtCount, EVT_COUNT_INTERVAL, 1);
```

```
        SetInputMode (evtCount, EVT_COUNT_REPS, 1);
}

int CreateHistogram(double *results,int n)
{

static int allocated=0;
double  max, min;
int  imax, imin, i, intervals;

if (allocated) {
  free(axisArray);
  free(histogramArray);
  }
MaxMin1D (results, n, &max, &i, &min, &i);
imin = RoundRealToNearestInteger(min);
imax = RoundRealToNearestInteger(max);
intervals = imax-imin+3;
axisArray=malloc(intervals*sizeof(double));       /* Allocate memory as needed */
histogramArray=malloc(intervals*sizeof(int));

Histogram(results,n,imin-1.5,imax+1.5,histogramArray,axisArray,intervals);

if (allocated) DeleteGraphPlot (evtCount, EVT_COUNT_GRAPH,-1,VAL_DELAYED_DRAW);
allocated = 1;

PlotXY (evtCount, EVT_COUNT_GRAPH, axisArray, histogramArray, intervals,
        VAL_DOUBLE, VAL_INTEGER, VAL_VERTICAL_BAR,
        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);

return(intervals);

}
```