Radioactive Decay

June 4, 2018

1 Introduction

In this two-week long lab, which will require a long lab write-up, you will build the data acquisition (DAQ) unit for a Geiger counter which you will use to count decays of a radioactive element. You will compare the data you collect using your custom DAQ to compare with the theoretical expectations for Poisson and Gaussian distributed random variables.

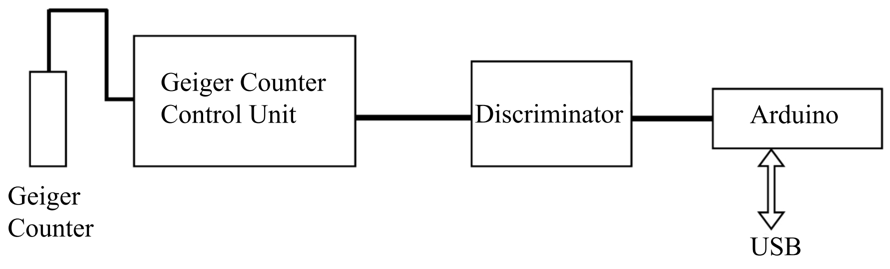


Figure 1: An overview of the lab setup. The Geiger counter and control unit are provided. You will build the discriminator stage and program an Arduino to complete the DAQ.

An overview of the lab setup is shown in Fig. 1. Once properly setup, the Geiger counter will produce a pulse like the one shown in Fig. 2 each time ionizing radiation passes through the chamber. The scope trace is AC-coupled, and therefore does not show the 5 volt offset (called a pedestal) that is present at the output. The first stage of the discriminator removes this pedestal with a high-pass filter. The subsequent stages convert these pulses above threshold into 5 volt digitized TTL pulses. Once connected to the Arduino, the DAQ unit receives a single TTL digital pulse on an input pin for each incident ionizing particle. The DAQ simply counts these pulses during a fixed time interval and reports the results over the serial connection. You'll build the discriminator, the DAQ, collect the data, analyze it using scientific python, and report everything in a long writeup.

2 Precautions

Precautions with the Geiger counter:

- Leave the cable from the Geiger counter controller to the Geiger counter in place *at all times.* This carries voltages of approximately 1000 volts. If you leave the cable in place, nothing can be inadvertently plugged in (including fingers!)

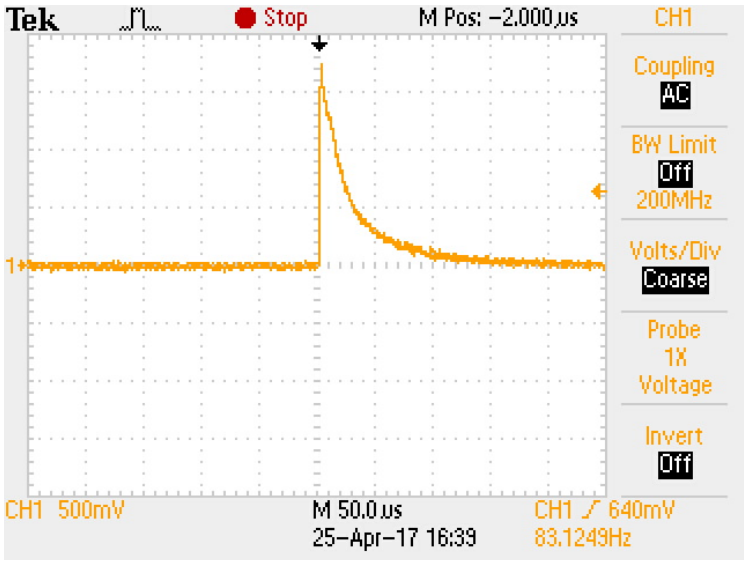


Figure 2: Typical pulse from the Geiger counter

- Leave the Geiger tube in its holder. It has a thin front window which is easily broken.

Precautions with the radioactive source:

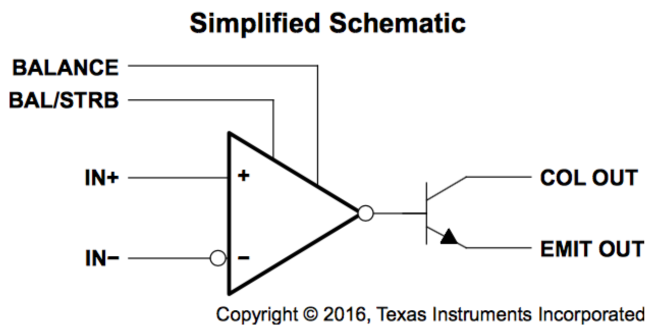
- Don't touch the source.
- Leave the source in the tray at all times. The TA will provide the sources and handle moving them from place to place.
- Radiation falls off as $1/r^2$. So minimize your time near sources and maximize your distance from them.
- Further information on radiation safety can be found in Melissinos and Napolitano.

3 AC-coupled Schmitt Trigger

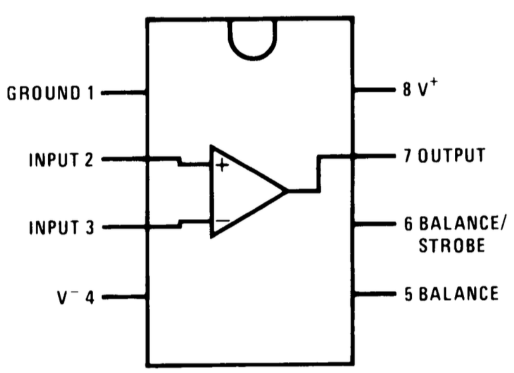
The first stage of the DAQ is a Schmitt trigger discriminator based on the LM311 (See Fig. 3) that you will build yourself. The LM311 is intended for TTL (5 V) logic and can be driven with a single-ended voltage all the way down to 5 V. This makes it an ideal component to use with our 5 V Arduino Uno. We can power the LM311 from the Arduino 5 V supply and send it's output directly to the Arduino. There is no need to get out the bench-top DC supplies!

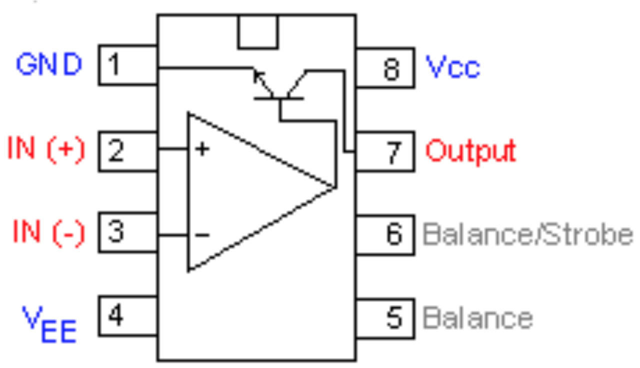
The LM311 is an open-emitter open-collector comparator, that is, an op-amp driving the base of an NPN diode with both the collector and emitter available as outputs. Typically the emitter of the output transistor is tied to ground and the collector output is therefore either grounded or high-impedance. For this reason, a pull-up resistor is generally needed at the the collector output, so that the output is either V_{CC} or ground, depending on which of the two inputs is largest.

The discriminator circuit you will be building is shown in Fig. 4. The input to the discriminator is AC coupled by R_1 and C_1, which is needed to remove the pedestal (DC offset) present at the Geiger counter output. This input is compared to a threshold predominantly set by the voltage divider R_2 and R_3. When the input signal (at the *inverting* input) exceeds this threshold, the base of the transistor is driven high (note the NOT) and so the output is pulled to ground. When the



(a)





(b) (c)

Figure 3: The LM311 (a) schematic from datasheet, and in 8-pin PDIP as (b) conventionally and somewhat deceptively shown, and (c) with the output transistor explicitly shown. The version (b) is still slightly deceptive, as it leaves out the not operation between the op-amp and the transistor base.

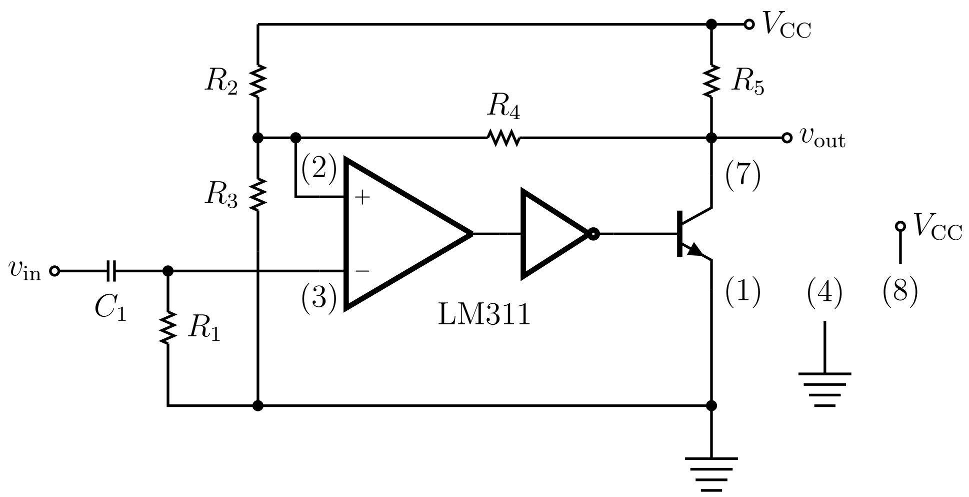


Figure 4: An AC-coupled inverting Schmitt trigger based on the LM311. The LM311 IC provides both the operational amplifier and the transistor. The numbers in parenthesis are pin numbers.

input signal is below the threshold, the base of the transistor is driven low, and the output is at V_{CC}, due to pull-up resistor R_5. The feedback resistor R_4 add hysteresis to the circuit, providing stability by pulling the threshold slightly higher once the output is high, and slightly lower once the output is low.

Build the circuit shown in Fig. 4 using $R_1 = 4.7$ kΩ, $C_1 = 10$ nF, $R_2 = 8.2$ kΩ, $R_3 = 1.1$ kΩ, $R_4 = 82$ kΩ, $R_5 = 1.1$ kΩ.

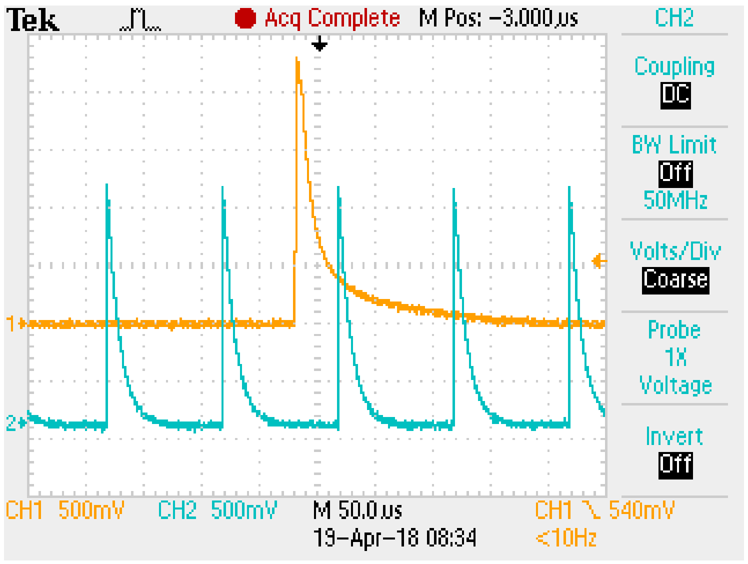


Figure 5: Side-by-side comparison of an actual Geiger output with the exponential decay function you will use to debug your data acquisition system.

For testing, you will need a reliable test signal that resembles the Geiger counter output. Use your Tektronix AFG1022 function generator, and select the "ExpDecay" from Arb/Others/Built-in/Maths. Set the frequency to 1 kHz and the amplitude to 2 V peak-to-peak. A side-by-side comparison of the exponential decay function and actual output from the Geiger counter is shown in Fig. 5. Typical output from the discriminator is shown in Fig. 6. Once your circuit is working properly, it is amusing to remove the feedback resistor R_4 and observe the resulting instability, as shown in Fig. 7.

4 Arduino Scalar

The next step in your DAQ is Arduino scalar. The scalar, or counter, will simply count the number of discriminator pulses that arrive during an adjustable time interval. These sums, and the duration of the time interval, will be displayed on the serial interface for your to analyze offline.

The output of your discriminator circuit is easily handled by an Arduino microprocessor, as it has already been digitized at the 5 V logic level used by your Arduino Uno. The digital pulses have a width of tens of microseconds, easily handled by an interrupt driven Arduino!

To get you started, an example sketch, "Interrupts", is available on the course website:

```
const byte ledPin = LED_BUILTIN;
const byte interruptPin = 2;
volatile byte state = LOW;
```

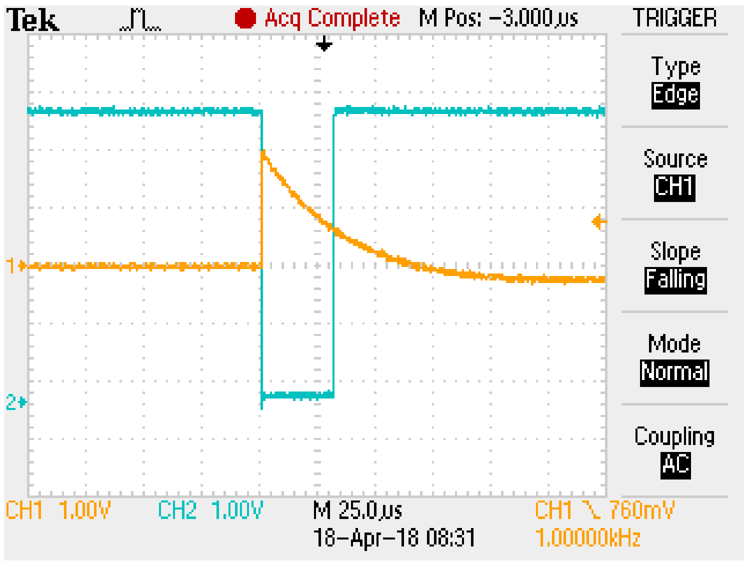
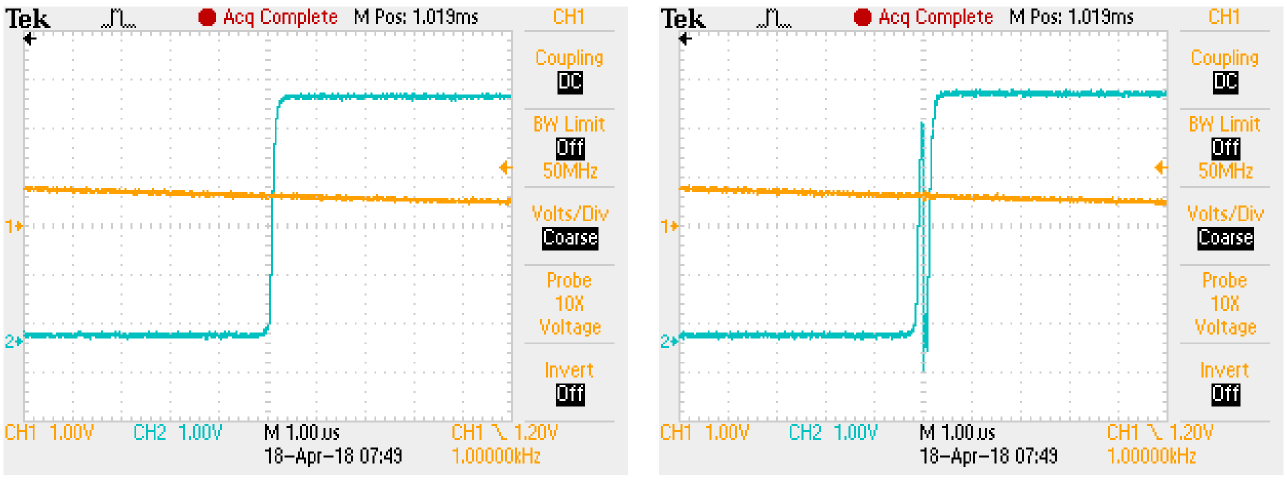


Figure 6: Typical output v_{out} of the discriminator circuit (at pin 7 of the LM311) compared to the input v_{in} (at pin3 of the LM311).



(a) (b)

Figure 7: Single trace for the rising edge of the discriminator circuit (a) as designed, and (b) with feedback resistor R_4 removed from the circuit. Without feedback, there is no hysteresis, and the output is bouncy. Once you have your DAQ working, you can check what removing this resistor would do to the stability and accuracy of your rate measurement. Spoiler alert: you need it!

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, RISING);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

This simple sketch sets up an interrupt service routine (`blink`) to handle interrupts on the rising edge of transition on digital pin 2. Every rising edge will cause the led to change state. If you connect your discriminant output to the Arduino digital pin 2, and turn the frequency of the function generator down to 1 Hz, you should observe the led to blink at 1 Hz.

Once you have this working, you should modify the blink interrupt service routine (suitably renamed!) to

- increment a count by one for every RISING (or FALLING) edge.

And modify the `loop()` function to do the following every time it is called:

- measure the current time in microseconds, via `micros()`.
- set the counter to zero.
- delay for a set time interval (e.g. 1 second: `delay(1000);`
- save the counter value
- measure the current time in microseconds.
- output the duration and count for this loop over the serial interface.

Technically, you should warn the compiler that the `count` global variable is effective by an ISR by using the volatile keyword, e.g.:

```
volatile int count = 0;
```

Once you have the counter working, you should verify that it is functioning properly by comparing the rate you measure with the Arduino to the rate your are sending with the function generator.

5 The Geiger Counter

On the Geiger Controller, start with the HV set to zero at both the analog knob and the HV on-off switch. Turn on the device and set the mode to"Test". In this mode, the counter will be incremented at a fixed rate of 60 Hz. While in this mode, play around with the "Count", "Reset", and "Lab-Chron" analog timer reset dial until you understand how all of these features work.

Next check that you have a source in the second highest drawer of the Geiger Counter holder, asking the TA for help if this is not the case. Switch the mode back to "Use", which will now use the Geiger counter as input to the count. With the HV still off, place the controller in "Count" mode, and verify that the rate, with no HV, is zero. Now, with the controller still in "Count" mode, turn on the HV and turn the dial until you first see the counter incrementing. Turn up the voltage to the next interval of 50 volts (e.g. if it first starts incrementing at 730 volts, set the dial to 750 volts).

Tabulate the number of counts in a 10 seconds interval, twice for each voltage level, in 50 volt steps from your starting voltage up to 1000 volts. Do not exceed 1000 volts. From the data, select the start of the plateau region, where increasing the voltage by 50 volts raises the rate by less than 10%. With another measurement, confirm that the rate at this plateau is of order 100 Hz (this will vary from source to source).

Look at the "SCOPE" output of the Geiger counter controller on your oscilloscope, with AC coupling, and verify that you see output pulses similar to those in Fig. 2.

6 Connect the DAQ

Now connect the "SCOPE" output of the Geiger counter controller to the input of your discriminator circuit. Now check the output of your discriminator and verify that you have the expected TTL output pulse relative to the input geiger counter pulses, as in Fig. 8. With the shape of the digital

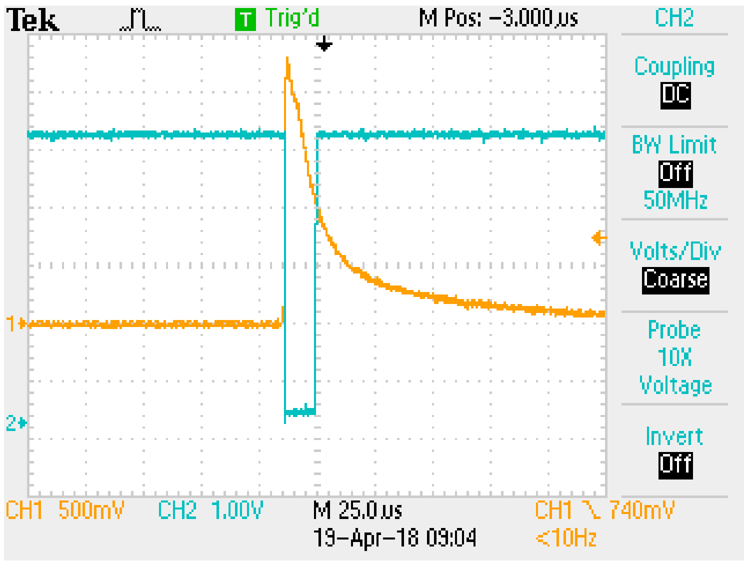


Figure 8: Output of the discriminant from actual Geiger counter ouput.

input pulse verified, plug this in as input to your Arduino DAQ. Verify that the rate you obtain with the Aruduino is within about 20% of the rate you estimated with the timer and counter features.

7 Collect and Analyze your Data

Now you are ready to collect the data you will use for your analysis. From your measured rate, estimate the sampling time interval needed to collect a mean number of events of 1,5, and 10. For each of these sampling time intervals, collect the count for 100 individual runs. Before leaving, verify that the mean value for each run is near the target mean value (but it does not need to exact, the idea is to have three samples *near* 1, 5, and 10 for comparison.) You could easily program your Arduino to take this average for you. If time permits, also collect data from 1000 individual runs at each value of τ.

8 Long Report

This lab report requires a long write-up. Your TA will provide a LaTeXtemplate for the report.

You should have the following sections:

- **Introduction:** Overview of experiment and its objective.
- **Methods:** Setup of the experiment, and calibration (e.g. Geiger HV setting) results, validation results (tests of basic functionality).
- **Monte Carlo Studies:** Test of your analysis procedure on simulated data (Monte Carlo).
- **Results:** Analysis of your experimental data. Compare collected data to appropriately chosen Poisson and Gaussian PDFs.
- **Conclusion:** Top-level summary of your results and what conclusions we can draw from them.
- **Appendix:** Code listing.