Calibration -
How to Measure Things Accurately

339 - Measurements Lab
Due February 19, 2018

February 2, 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

Calibration is the establishment of the relationship between a well known quantity, and an intrinsic property of some system. This relationship then allows the use of this system to make measurements in terms of the original well known quantity. Back when the meter was first defined, there was, and still is only one platinum-iridium bar which served as the standard. All other measurements of length had to derived by comparison with this unit. This done, individuals who have measuring instruments calibrated to the original unit can discuss measurements using their instrument with one another, confident that with in the error of their instrument they agree about the values they measure. Today where ever possible units have been defined in terms of fundamental atomic properties, and the speed of light. In the particular case of this lab, the DAC provides voltages on the analog output which is controlled by the 8 bit number applied to the digital input controlled by software. Similarly the ADCs allow software to read 10 bit numbers which corresponts to a particular voltage applied to the analogue input of the device. These numbers are probably useless unless they can be converted into standard units.

To determine the relationship between the voltage upon the output of a DAC and the 8 bit number applied to its input, a set of numbers should be sent, and the response of the output measured with a voltmeter. The voltage reponse can then be plotted as a function of the number sent, and a decision made about the nature of the relationship. If the relationship is a simple function, then the conversion can be obtained quite easily. For example, if the relationship were linear, that is

$$V_{DAC} = V_0 + GN_{DAC} \tag{1}$$

where V_{DAC} is the voltage at the output in reponse to the number N_{DAC} applied to the input port. V_0 and G would be properties of the DAC. If V_0 and G can be determined then the relationship can be inverted, and used to set the output of the DAC to a desired voltage. In the case of a linear relationship we can use linear regression to obtain the parameters.

Alternatively, they can be downloaded from my courses in a compressed .tar file Calibration_2018.tar, which can be extracted to your group folder.

[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

2 Linear Regression

If we have a linear relationship between two variables,

$$y = a + bx \tag{2}$$

and a set of N measurements $((x1, y1, \sigma 1), (x2, y2, \sigma 2), ...(xN, yN, \sigma N))$, then we can establish the parameters a and b which minimize following quantity,

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{a + bx_i - y_i}{\sigma_i^2} \right)^2 \tag{3}$$

If χ^2 is minimized, the following will be true,

$$\frac{\partial \chi^2}{\partial a} = \sum_{i=1}^{N} \left(\frac{2a + 2bx_i - 2y_i}{\sigma_i^2} \right) = 0 \tag{4}$$

and

$$\frac{\partial \chi^2}{\partial b} = \sum_{i=1}^{N} \left(\frac{2bx_i^2 + 2ax_i - 2x_i y_i}{\sigma_i^2} \right) = 0 \tag{5}$$

Expanding the summations in eq(4) and eq(5),

$$a \sum_{i=1}^{N} \frac{1}{\sigma_i^2} + b \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} - \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} = 0 \tag{6}$$

and

$$b \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} + a \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} - \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} = 0 \tag{7}$$

Making the following substitutions,

$$\alpha = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \qquad \beta = \sum_{i-1}^{N} \frac{x_i}{\sigma_i^2}$$

$$\gamma = \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} \qquad \lambda = \sum_{i-1}^{N} \frac{x_i^2}{\sigma_i^2}$$

$$\rho = \sum_{i-1}^{N} \frac{x_i y_i}{\sigma_i^2}$$

The equations may be combined and written

$$\begin{bmatrix} \alpha & \beta \\ \beta & \lambda \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \gamma \\ \rho \end{bmatrix} \tag{8}$$

Multiplying Eqn 8 with the inverse of the square matrix,

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \alpha & \beta \\ \beta & \lambda \end{bmatrix}^{-1} \begin{bmatrix} \gamma \\ \rho \end{bmatrix} = \begin{bmatrix} M_{(1,1)} & M_{(1,2)} \\ M_{(2,1)} & M_{(2,2)} \end{bmatrix} \begin{bmatrix} \gamma \\ \rho \end{bmatrix} \tag{9}$$

To obtain the uncertainties on the parameters, we use standard error propagation. Reading off the result for parameter a from Eqn. 9,

$$a = M_{(1,1)} \sum \frac{y_i}{\sigma_i^2} + M_{(1,2)} \sum \frac{x_i y_i}{\sigma_i^2} \tag{10}$$

So the uncertainty on parameter a is determined by

$$\begin{aligned}
\sigma_a^2 &= \sum_{i=1}^{N} \left(\frac{\partial a}{\partial y_i}\right)^2 \sigma_i^2 && (11)\\
&= \sum_{i=1}^{N} \left(M_{(1,1)}\frac{1}{\sigma_i^2} + M_{(1,2)}\frac{x_i}{\sigma_i^2}\right)^2 \sigma_i^2 && (12)\\
&= \sum_{i=1}^{N} \left((M_{(1,1)})^2\frac{1}{\sigma_i^2} + (M_{(1,2)})^2\frac{x_i^2}{\sigma_i^2} + 2M_{(1,1)}M_{(1,2)}\frac{x_i^2}{\sigma_i^2}\right) && (13)\\
&= (M_{(1,1)})\sum_{i=1}^{N}\frac{1}{\sigma_i^2} + (M_{(1,2)})^2\sum_{i=1}^{N}\frac{x_i^2}{\sigma_i^2} + 2M_{(1,1)}M_{(1,2)}\sum_{i=1}^{N}\frac{x_i^2}{\sigma_i^2} && (14)
\end{aligned}$$

Recognizing the appropriate substitutions for the summations and grouping preemptively,

$$\sigma_a^2 = M_{(1,1)}\left(M_{(1,1)}\alpha + M_{(1,2)}\beta\right) + M_{(1,2)}\left(M_{(1,1)}\beta + M_{(1,2)}\lambda\right) \tag{15}$$

But, since M is the inverse of our original matrix,

$$\begin{bmatrix} M_{(1,1)} & M_{(1,2)} \\ M_{(2,1)} & M_{(2,2)} \end{bmatrix}\begin{bmatrix} \alpha & \beta \\ \beta & \lambda \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{16}$$

Eqn. 15 simplifies to

$$\sigma_a^2 = M_{(1,1)} \tag{17}$$

Similarly,

$$b = M_{(2,1)}\sum\frac{y_i}{\sigma_i^2} + M_{(2,2)}\sum\frac{x_i y_i}{\sigma_i^2} \tag{18}$$

So the uncertainty on parameter b is determined by:

$$\begin{aligned}
\sigma_b^2 &= \sum_{i=1}^{N} \left(\frac{\partial a}{\partial y_i}\right)^2 \sigma_i^2 && (19)\\
&= \sum_{i=1}^{N} \left(M_{(2,1)}\frac{1}{\sigma_i^2} + M_{(2,2)}\frac{x_i}{\sigma_i^2}\right)^2 \sigma_i^2 && (20)\\
&\ldots && (21)\\
&\ldots && (22)\\
&= M_{(2,1)}\left(M_{(2,1)}\alpha + M_{(2,2)}\beta\right) + M_{(2,2)}\left(M_{(2,1)}\beta + M_{(2,2)}\lambda\right) && (23)\\
&= M_{(2,2)} && (24)
\end{aligned}$$

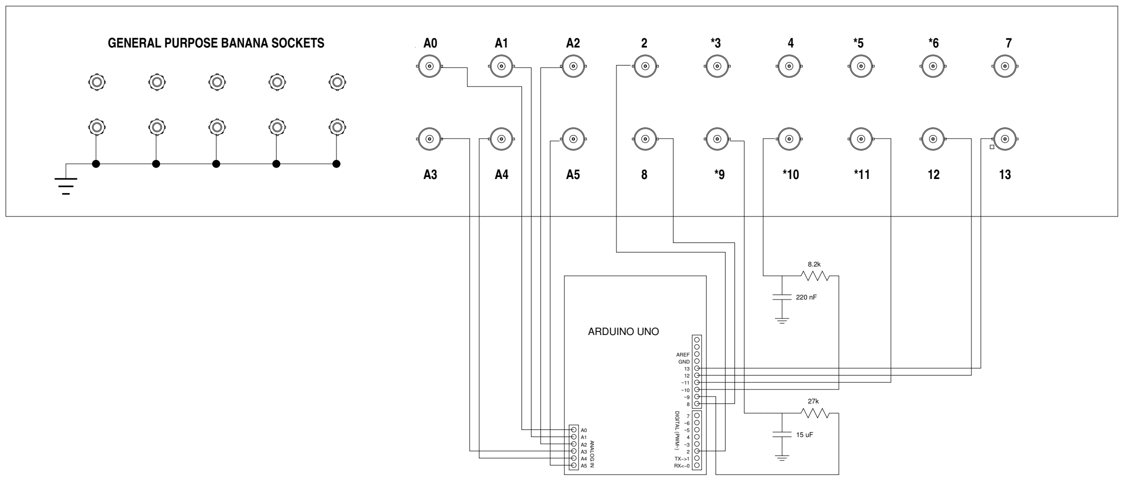

Figure 1: The arduino front panel. Note that Pin 8 has been replaced by the output of a true DAC (see Section 8).

3 The Hardware

Figure 1. shows the front panel of a typical Arduino as we have installed them. The digital to input / outputs are located at the far right of the panel. Earlier in the text we have refered to the digital to analog converters (DAC). The Arduino Uno does not have any true DACs. It has some digital ports which are capable of pulse width modulation (PWM). These ports are identified by a tilde ( ) on the Arduino board, and by and asterisk () on the front panel (since the label maker doesnt have a tilde character) In order to fake a DAC, PWM outputs 9 and 10 have been wired up with a low-pass filter. PWM, as the name suggests gives an rectangular wave of with a duty cycle which varies according to the value requested the minimum value with always be zero, maximum will always be 5V, 90% of maximum value will result in 90% duty cycle. PWM outputs are 8-bit, that is legal values are in the range 0 to 255. The low-pass filter on PWM 9 has a nominal -3dB frequency of 0.4 Hz which eliminates most of the ripple, but which reacts very slowly. It is most suited to calibration of the ADC and for applications where a very stable volage is required. Not, this outout has a high output impedance and will require buffering to drive loads. PWM 10 has a nominal -3dB frequency of 70 Hz which passes far more ripple. This output also has a moderately high output impedance.

The analogue to digital converter inputs are the six leftmost BNC connectors, there are six channels, 0 - 5. ADCs are 10 bit, that is they return numbers in the range 0 to 1023, all voltages are limited to 0 to 5V.

4 The Measurements

The first exercise will be to calibrate the pseudo DAC, Arduino pin 9. An example sketch for the **Arduino**, `Calibration.ino` is provided. It may be used as is. If you open a serial monitor from the **Arduino** IDE it is somewhat self documenting. It is suggested that you connect PWM output 9 to the DMM and characterize the relation between number sent and measured voltage. How will you determine the error on this measurement? You should plot the response curve, and determine what the relationship is. If you are confident the relationship is linear you may use linear regression to characterize the DACs. If you correctly characterize your DACs you will be able to set the voltage of a given DAC to within the precision of the DAC. What is the precision of the DAC, what is the origin of this precision? Characterize the ripple on the DACs, is it amplitude dependent? Modify the sketch so that the period of the PWM is not modified in the `setup()` function. How does this affect the ripple? Can you estimate the properties of the low-pass filter from the ripple? (Hint look at the PWM outputs which dont have low-pass filters attached on the oscilloscope).

If you have successfully calibrated the DAC, then you now have a precision voltage source, you can then use this to provide stimulus to the ADC channels by connecting the output to the analog input of interest, then recording the resulting numbers. You should do this with a program, rather than any manual activity. To this end, you have a variety of options. You could use `Calibration.ino` as a starting point and modify it to send numbers to PWM 9 and send the results out using `Serial.print()` and/or `Serial.println()`. To bring these numbers into **Python**, it shouldnt be too big a challenge to modify one of the previous interfaces to the **Arduino** and just grab the values using `getResp()`. For those of you who do not feel confident about this approach, take a look at the combination of Calibration Python.ino and calibration.py which implements a server on the **Arduino** which will work with the **Python** client to provide a variety of canned functions which are documented later. In all cases, be aware that PWM 9 reacts slowly, you will need to wait for the voltage to settle to equilibrium before it matches your calibrated expectation value.

Verify that all six ADC channels work, and have similar characteristics. Remember that the value returned is only meaningful if you have a voltage applied to the input, otherwise you are just measuring the voltage stored in the capacitance of the input (ADCs have high input impedance.)

When analysing the data, it will be useful to examine the residual of your fit, that is $y_i f(x_i)$, this should be lie on the x-axis, with points scattered consistent with their uncertainty. It is not valid to claim that a fit is good because the fit-line looks identical to the data if the range of y_i is much greater than σ_i. Do not be surprised if your analysis reveals inaccuracies in the measurements reported by your voltmeter.

The final exercise is fairly open ended. You are to write a software function generator using the **Arduino** for output. A basic generator should be capable of generating the basic functions sinewave, triangle, squarewave and sawtooth. These should have user specified amplitude and frequency. What limitations are placed upon these parameters? You should discuss in your report what your specifica- tions on the function generator are, you will be asked to demonstrate your function generator, and will be marked upon how realistic your specifications were. The fact that the DAC is actually a PWM output with a low-pass filter makes this problem rather more challenging, since the raw signal sent to the pin will be undergo a frequency dependent distortion of both amplitude and phase. It will be possible to correct for this if you correctly characterize the behavior of the filter.

You will recall in the Introduction Exercise there was a signal, which when correctly transformed by the low-pass filter function, became a triangle wave. Go back and examine the `Low_Pass_Filter_by_FFT()` function. You could modify the code so the rather than emulate the action of a low pass filter, it would perform the inverse operation so as to create a signal which, when passed through the physical low-pass filter, recreates the original signal.

There are a several ways you could characterize the low-pass filter on PWM 10: in the frequency domain by measuring the response to sine waves at different frequencies, or in the time domain by measuring the transient response. The function `analogWriteReadVector` in the canned routines can be used for both these approaches.

If you go the route of programming the **Arduino** directly you can generate the various waves by superposition of sinewaves with amplitude and phase correction. If you go with the canned routines, you can generate the desired wave and perform the fourier-space correction before sending the wave to the **Arduino**.

5 Low-Pass Filter

Recall a R-C low-pass filter affects both amplitude

$$V_{out} = \frac{1}{\sqrt{1+\left(\frac{f}{f_c}\right)^2}} V_{in} \qquad (25)$$

where $f_c = \frac{1}{2\pi RC}$ and V_{in} and V_{out} are the input and output amplitudes of a sinewave of frequency f. The phase of the output lags the input by ϕ,

$$tan\phi = \frac{f}{f_c} \tag{26}$$

6 Software

You are provided with **Python** code implementing a communication protocol with the **Arduino**, called `arduino.py`. The arduino class implements the following functions which might be of interest.

- `analogRead` - this performs a single reading from an ADC.
- `analogWrite` - this performs a single write to a PWM capable digital pin.
- `analogReadVector` - returns a vector of readings from an ADC
- `analogWriteVector` - sends a vector to PWM capable digital pin and loops through the vector until a new command is sent.
- `analogWriteReadVector` - like `analogWriteVector`, except that after the requested number of iterations through the vector, it returns the ADC values read on the last loop.
- `out_buffer_length` - returns the maximum length of a vector sent using `analogWriteVector` and `analogReadWriteVector`.
- `in_buffer_length` - returns the maximum length of a vector read using `analogReadVector`.
- `sampling_time` - returns the duration in seconds of a single sample in the vector base functions.

7 Resources

The `ardiuno` class relies upon a sketch, `Calibration_Python.ino` being loaded on the **Arduino**.

Also provided are two example usages of the class: `demo_arduino_simple.py` and `demo_arduino_complex.py`, which should give you some ideas for implementing the function generator. All provided functions can be found in `/mnt/resources/339/Calibration_2018`. Copy all of these to your group folder using:

```
cp -rf /mnt/resources/339/Calibration_2018 /mnt/home/phys339-nn
```

in the terminal.

8 Optional reading/usage - DAC

Recently, output Pin 8 has been modified by the inclusion of a **real** DAC, the MCP4725. This can be used to output **actual** analog signals if you so desire (hint - this may be useful during your final project and/or for your function generator). For those of you brave enough to attempt this feat (really, it's not that tricky though!) a nice driver library has already been written, which can be downloaded from:

`https://github.com/adafruit/Adafruit_MCP4725`

and installed where your arduino libraries go, most likely `/mnt/home/yourUserName/libraries/`, but double check this in the Arduino software.