[[1]](#footnote-1)

Lab 4: Sensor Noise and Quantization Noise Measurements

Jonathan Sumner, Diego Lopez, Rachel DuBois, and Argenis Aquino, Department of Engineering Technology, Rochester Institute of Technology

*Abstract*— This paper presents an approach for characterizing and reducing noise in an LM61 temperature sensor measurement system interfaced with a 10‐bit Arduino ADC. First, raw sensor noise is measured by examining chatter across ADC thresholds. Next, the total noise is reduced via oversampling and averaging. Finally, additional noise (“dither”) is injected to randomize quantization boundaries, allowing the Arduino to resolve signals below its native LSB. Experimental results demonstrate a 3‐bit improvement in effective resolution and an 18 dB increase in signal‐to‐noise ratio (SNR) when oversampling and dithering are combined.

# INTRODUCTION

A

ccurate analog‐to‐digital conversion (ADC) is essential in many sensing applications, particularly temperature measurement. The Arduino Uno’s 10‐bit ADC, with a 5 V reference, generates a least significant bit (LSB) of about 4.88 mV and a quantization‐noise standard deviation near 0.29 LSB. In cases where the sensor noise is small, the ADC output can remain stuck around individual code boundaries, producing “chatter” rather than uniformly traversing multiple codes. The LM61 temperature sensor, known for its low noise, can exhibit this behavior when it experiences slow temperature variations. Although oversampling and averaging can reduce noise by a factor of the square root of the number of samples, this theoretical limit only holds if enough randomness exists in the raw input signal. In scenarios where inherent noise is insufficient, adding a low‐level dithering signal helps the ADC readings cross more code boundaries uniformly, making simple averaging more effective. This paper details measurements of LM61 sensor noise, quantification of oversampling improvements, and the subsequent gain from injecting a dither signal. Section II outlines the sensor hardware and data‐acquisition procedures, including how drift removal and noise metrics are computed. Section III presents measured results for raw sensor noise, oversampling alone, and dithering plus oversampling, with final calculations of the system’s effective bit depth. Section IV concludes by summarizing the benefits of combining dithering and oversampling to enhance ADC resolution.

# Methodology

## Assessing Sensor Noise, Accuracy, and Signal-to-Noise Ratio (Section 1A)

In this section, the LM61 temperature sensor was connected to the Arduino Uno’s +5V, GND, and A0 ports as shown in **Fig. 1**. The Arduino was programmed with the provided *DSP\_Lab04\_CodeBase.ino*, which enabled the Arduino to acquire a series of temperature samples from the LM61 temperature sensor.

After programming, the sensor was briefly warmed by applying light pressure (using two fingers) for 10 seconds. To capture the sensor’s subsequent colling data in MATLAB, the following command was executed from the Command Window (adjusting ‘ComPort’ as necessary): ***data = CaptureArduinoData(‘ComPort’, 3, ‘BaudRate’, 115200, ‘NumActivePlots’,1)***. This command launched real-time data collection, which was saved as **data-1a.mat**.

To facilitate sensor‐noise estimation, we specifically looked for segments of data that contained multiple *chatter bursts*, the repeated crossing of adjacent ADC code bins, since a slowly drifting sensor often exhibits these bursts. A suitable data subset was then chosen, and its plot appears later in the *Results* section.

Next, we estimated the sensor noise by combining the known quantization noise standard deviation (0.29 LSB) with measured chatter characteristics. A polynomial drift‐removal step was also applied to this chosen data segment, using MATLAB’s polyfit and polyval functions to generate a first‐order fit and subtract it from the raw data. Finally, the drift‐removed signal’s standard deviation was computed, providing a direct comparison to the predicted noise levels.

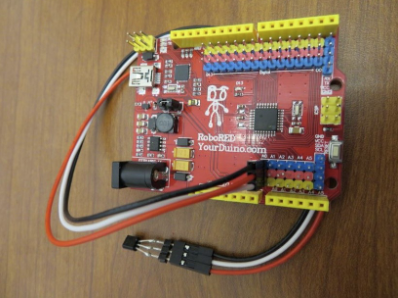
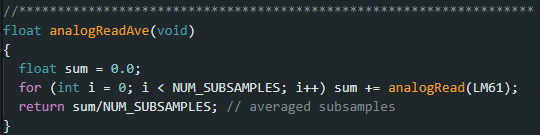
**

Fig. : LM61 connected to the Arduino via the A0 Port, GND, and +5V

## Oversample and Averaging to Reduce System Noise (Section 1B)

In this section, oversampling and averaging were implemented to reduce noise in the LM61 temperature readings. The Arduino was configured similarly to Section 1A, but the line ***sample = analogRead(LM61)*** was commented out, and the line ***sample = analogReadAve()*** was uncommented. The ***analogReadAve()*** function loops *NUM\_SUBSAMPLES* times, sums the sensor ADC readings, and divides by that number of subsamples:

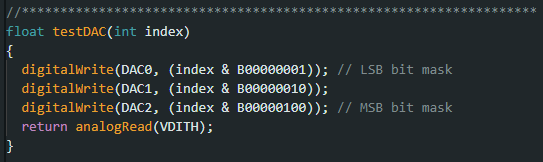


After uploading the modified code, data were recorded using the same ***CaptureArduinoData*** procedure described earlier, resulting in a file named **data-1b.mat**. A segmented portion of the collected data—encompassing a few distinct “stair‐step” levels in the ADC readings—was then selected for further analysis. To account for gradual temperature changes, a first‐order polynomial drift was fit to the data using MATLAB’s polyfit and polyval functions, and this drift estimate was subtracted from the measured samples. The residual error was then plotted, and its standard deviation was computed to verify that oversampling and averaging had successfully reduced noise compared to the raw sensor output.

## Adding Dithering Noise to Enable Oversampled Averaging (Section 1C)

Here, **dithering** was introduced to further enhance the performance of oversampling and averaging. A small digital‐to‐analog converter (DAC) circuit was built on a protoboard, as shown in **Fig. 2**, with nominal resistor values (e.g., 47 Ω for dither injection).

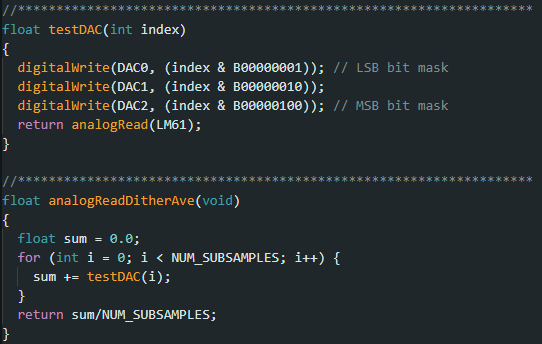
### DAC Testing (Ramp and Random Data)

Initially, the temperature sensor was **disconnected**, and the function call ***sample = testDAC(tick)*** was enabled in the Arduino code. The global variable ***NUM\_SAMPLES*** was set to **256**. Here, ***testDAC(tick)*** outputs a ramp signal to the DAC (by incrementing **tick**), which is read back on analog pin A1:   
This code was uploaded, and data were captured to **data-1c-testDAC.mat**, from which a histogram and cumulative density function (CDF) were generated. The typical error of this ramp output was then computed.

Next the code was modified so that ***sample = testDAC(random(256))*** replaced the tick ramp with a random sequence of 256 values. A second data set was collected and saved as **data-1c-random.mat**, and the random-signal error was similarly analyzed with a plot, histogram, and CDF.

### Dither Voltage Generator Hardware Description

After confirming that the DAC circuit functioned correctly, the 2.2kΩ resistor used for testing was replaced with the 47Ω resistor to generate a small dithering noise. In the Arduino code, ***sample = testDAC(random(256))*** was commented out, and ***sample = analogReadDitherAve()*** was uncommented.

The custom ***analogReadDitherAve*** function combines the concepts of ***analogReadAve*** and ***testDAC***, providing dither for each oversampled measurement of the LM61 sensor: 

The sensor was then reconnected, and a 10‐second warming procedure was repeated before capturing new data with MATLAB, saved as **data-1c-dither.mat**. Finally, drift removal (using a polynomial fit) and standard deviation calculations were performed on this new data set, and a histogram and CDF of the resulting noise were plotted.

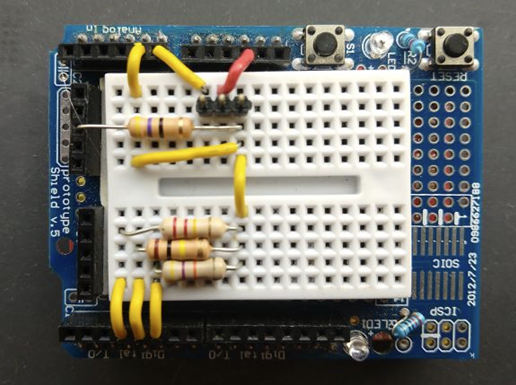


Fig. : DAC Hardware Setup with 47Ω, 220kΩ, 100kΩ, and 47kΩ resistors

## Sensor Noise Levels for Different Processing Methods (Table 1)

Table 1 consolidates both **measured** and **theoretical** noise levels for each processing method—raw, oversampled, and oversampled with dither. The entries reflect the **sensor noise** (from Section 1A), the **quantization noise** (approximately 0.29 LSB), and any **dither noise** introduced by the DAC circuit, scaled by the ratio of resistor values (e.g., 47 Ω / 2,200 Ω). These noise contributions are then combined via a root‐sum‐square to yield the **total input noise**, which is subsequently reduced by a factor of the square of 160​ under ideal oversampling. Table 1 compares this **theoretical** noise reduction against the **measured** standard deviations (Sections 1B and 1C), enabling a direct assessment of the effectiveness of simple oversampling versus oversampling with dithering.

## Effective ADC Bit Depth from Dithering Noise and Averaging (Section 1D)

Finally, we quantify the improvement in **effective ADC resolution** resulting from the combination of oversampling, dithering, and averaging. The **improvement ratio** is defined as the ratio of the standard deviation of the total noise (Section 1A) divided by the standard deviation is the residual error after oversampling and dithering (Section 1C). The number of additional bits gained is the base-2 logarithm of the improvement ratio. Adding the additional bits gained to the Arduino’s nominal **10-bit resolution** yields the **effective bit depth** of the enhanced ADC measurements. This final calculation confirms how much dither plus oversampling can push the performance beyond the baseline 10-bit limit.

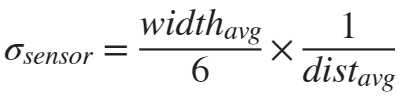
# Results

## Assessing Sensor Noise, Accuracy, and Signal-to-Noise Ratio (Section 1A)

In this experiment, an LM61 temperature sensor was held for 10 seconds, and the raw output data were recorded. As shown in **Fig. 3**, the time-series data were segmented to identify chatter start and end points. The average chatter width was determined to be **53.6667**, and the estimated distance between chatter bursts was **193**.

From these values, the standard deviation of the sensor noise was computed using standard deviation of the sensor noise, see **(1)**, yielding **0.0463 dB**. Next, the **total noise** was estimated by combining the sensor noise and quantization noise according to **(2)**, resulting in **0.2937 dB**. To account for drift, we subtracted a drift estimate from the segmented data and recalculated the standard deviation, obtaining **0.3587 dB**. The drift-corrected data are illustrated in **Fig. 4**.

**(1)**



**(2)**



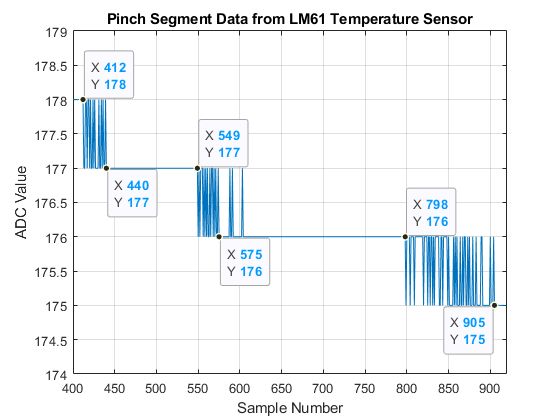


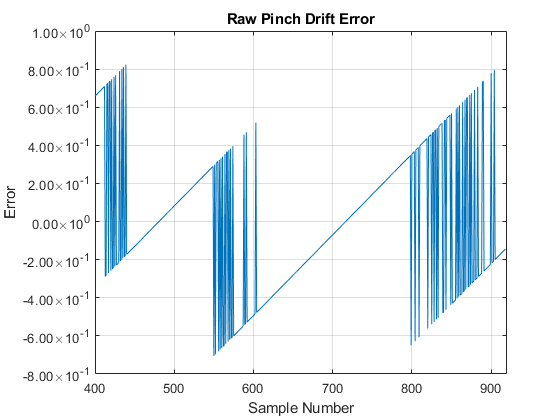
Fig. : Segmented data with data points set to chatter start and end points

Fig. : Plot of the raw pinch data with the drift removed

## Oversample and Averaging to Reduce System Noise (Section 1B)

Oversampling and averaging techniques were then applied to reduce noise in the LM61 temperature sensor readings. **Fig. 5** shows a segmented portion of the oversampled data with three stair steps, as well as an underlying temperature drift. After estimating and subtracting this drift from each ADC reading, the standard deviation of the resulting error was computed to be **0.1916 dB**. **Fig. 6** illustrates the residual error (“Pinch Drift Error”) obtained from 160 oversamples and averaging. This confirms that oversampling combined with drift removal significantly reduces the effective noise floor.

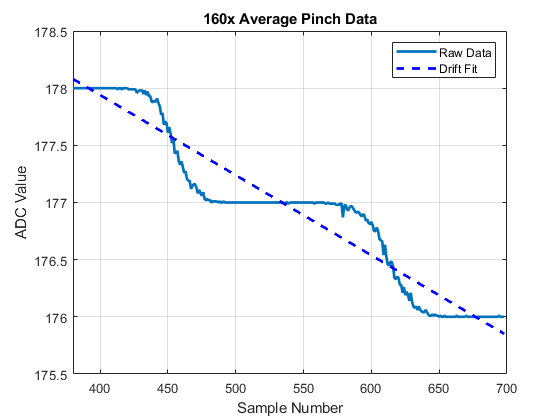


Fig. : Plot showing 3 stair steps from the oversampled data and the underlying temperature drift

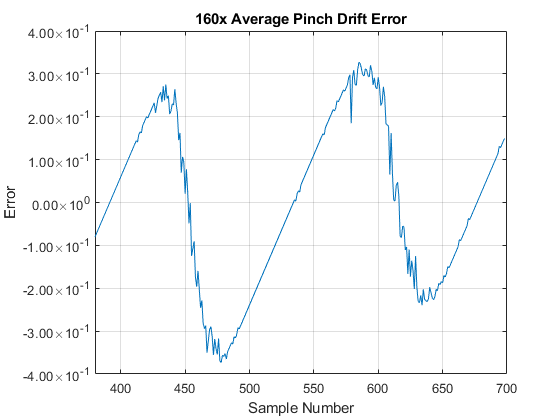


Fig. : Plot showing the Pinch Drift Error for the 160 oversamples and averaging

## Adding Dithering Noise to Enable Oversampled Averaging (Section 1C)

In the final set of tests, two 256‐sample data sets were collected using the Arduino’s testDAC() function. The first was a ramp data set generated by incrementing a variable tick. Because tick reset after every 7 counts. The second data set used random(256) to produce random samples. **Fig. 7** illustrates 50 of these 256 random samples, while **Figs. 8** and **9** present its histogram and CDF.

Analyzing these data sets, we computed standard deviations (beginning at the second index) for each distribution. The ramp data exhibited a typical error of **24.5545 dB**, whereas the random data yielded **25.4965 dB**. These results suggest that the random data distribution is slightly broader in spread, which aligns with expectations for uniformly distributed random samples compared to a controlled ramp.

The DAC was then injected with dither noise into the temperature measurement. **Fig. 10** plots the data with dither noise added to the system, **Fig. 11** shows the residual error signal with dither, and **Fig** **12** presents the histogram. With the addition of dither now in the system, the standard deviation of the residual error signal resulted in a standard deviation of **0.0223 dB**.

By taking the standard deviation of the residual error signal and applying it to the SNR equation, see **(3)**, the resulting SNR is **33.0339 dB**. Comparing this value to the oversampling (without dither) SNR, which resulted in a value of **14.3521** **dB**, an improvement of **18.6818 dB** is seen by injecting the dither noise. This improvement was caused by the dither noise randomizing the ADC quantization steps so that simple averaging can smooth out those quantization boundaries more effectively. The theoretical noise after dithering and oversampling was **0.0489 LSB** whereas the measured noise was **0.0223 LSB**, a significant difference than predicted.

Comparing no dither vs dither and averaging with 256 samples results in the same qualitative result as the 160 sample. The theoretical reduction in noise is **24.0824 dB** improvement. The experimental data show an improvement of **18.6818 dB** going from no dithering to dithering. Minor differences from 24 dB stem from real-world imperfections such as sensor drift, finite amplifier bandwidth, and slightly non-ideal dither amplitude.

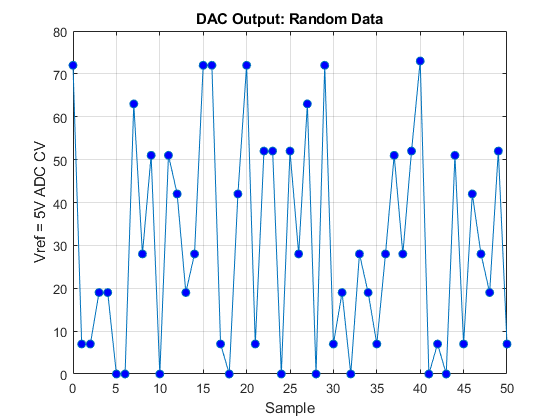


Fig. : Plot of random data collected from the testDAC function with random(256)

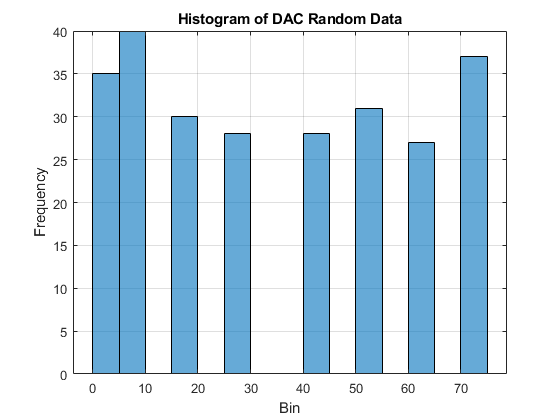


Fig. : Histogram plot showing the frequency of DAC values (bin)

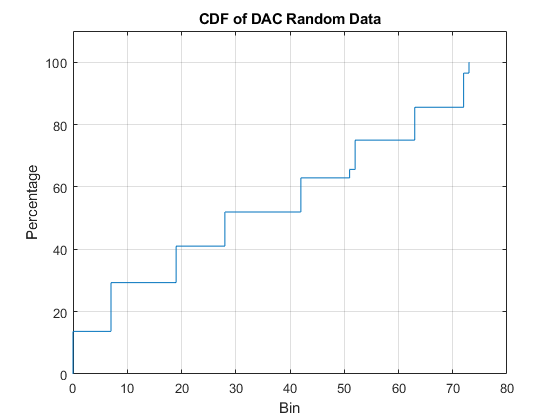


Fig. : CDF plot showing the percentage of samples with values up to each DAC output value (bin)

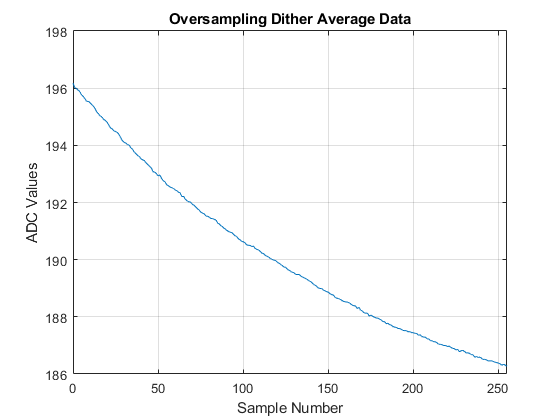


Fig. : Plot of oversampling and averaging with dither noise injected

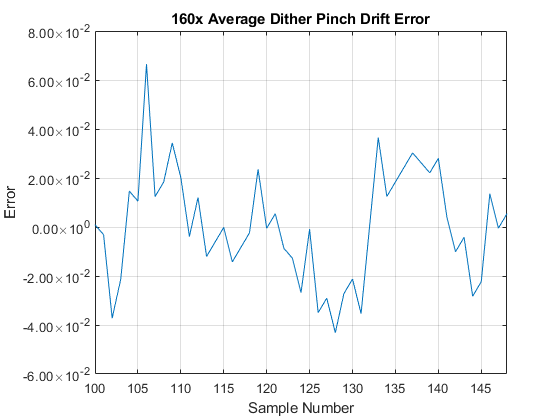


Fig. : Plot of oversampling and averaging with dither noise injected

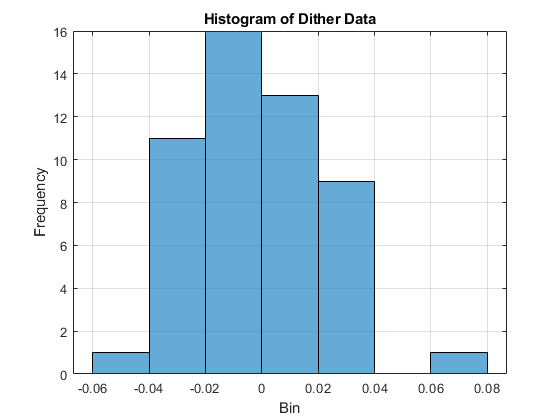


Fig. : Histogram plot showing the frequency of DAC values (bin)

**(3)**

## Sensor Noise Levels for Different Processing Methods (Table 1)

Table I consolidates the noise parameters obtained from Sections 1A through 1C. The “Raw Values” column corresponds to the baseline sensor acquisition (no averaging), while the “Oversampling and Averaging” column represents 160 oversampling without dithering. Finally, the “Dithering and Averaging” column includes an artificially injected noise signal (scaled by the ratio 47 Ω/2.2 kΩ) to further randomize the ADC quantization steps.

|  |  |  |  |
| --- | --- | --- | --- |
| Quantity | Raw Values (No Averaging) | Values after 160x Oversampling and Averaging (No Dithering) | Values after Dithering and Averaging (with Dithering) |
| 1 Code Value Signal | 1.00 | 1.00 | 1.00 |
| Sensor Noise SD | 0.0463 | 0.0463 | 0.0463 |
| Quant Noise SD | 0.29 | 0.29 | 0.29 |
| Dither Noise SD | 0.00 | 0.00 | 0.5447 |
| Total Input Noise | 0.2937 | 0.2937 | 0.6188 |
| Noise after averaging (theoretical) | N/A | 0.0232 | 0.0489 |
| Noise after averaging (measured) | N/A | 0.1916 | 0.0223 |

From the table, we see that the raw (unaveraged) sensor and quantization noise remain around **0.2937** LSB total. In the oversampling‐only case, theory predicts a reduction to **0.0232** LSB, though measured noise remains higher (**0.1916** LSB) due to non‐ideal factors such as sensor drift. With dithering added, the total input noise initially increases (**0.6188** LSB), but oversampling more effectively averages out the now‐randomized quantization boundaries, bringing the measured noise down to **0.0223** LSB, very close to the theoretical **0.0489** LSB once the dithering amplitude is accounted for.

## Effective ADC Bit Depth from Dithering Noise and Averaging (Section 1D)

The ratio of the raw total noise (0.2937 LSB) to the residual error with oversampling and dithering (0.0223 LSB) was found to be **13.1979**, corresponding to a **3.722-bit** improvement. By adding these extra bits to Arduino’s base 10‐bit resolution, the system achieves an **effective 13.722‐bit** conversion depth, closely matching the theoretical predictions. This result highlights the combined benefits of oversampling and dither in pushing beyond the nominal 10‐bit quantization limit.

Consequently, the temperature resolution of the final system is determined to be **0.37 mV**, which is significant improvement to the temperature resolution when the system was raw at **4.88 mV**.

# Conclusion

The LM61 sensor’s low intrinsic noise, combined with slow temperature changes, can result in minimal ADC code transitions and modest oversampling gains. Although 160× oversampling reduced noise from 0.294 LSB to 0.1916 LSB, this fell short of the ideal theoretical factor. However, the introduction of a small dithering voltage randomized the quantization steps enough to drive the noise floor to 0.0223 LSB, yielding an approximately 3.7‐bit improvement and boosting the Arduino’s 10‐bit ADC to a near 13.7‐bit effective resolution. These results underscore the synergy between dither and oversampling for low‐noise measurements, demonstrating that carefully added random noise can unlock higher precision in applications where sensors otherwise fail to produce sufficiently random ADC samples on their own.

1. [↑](#footnote-ref-1)