Converting analog data to binary

The real world is **analog**, a continuous stream of varying data.

Just look around you, beyond your computer or phone. There's an infinite amount of visual information. If you zoom into one part of your visual field, you can notice more and more details.

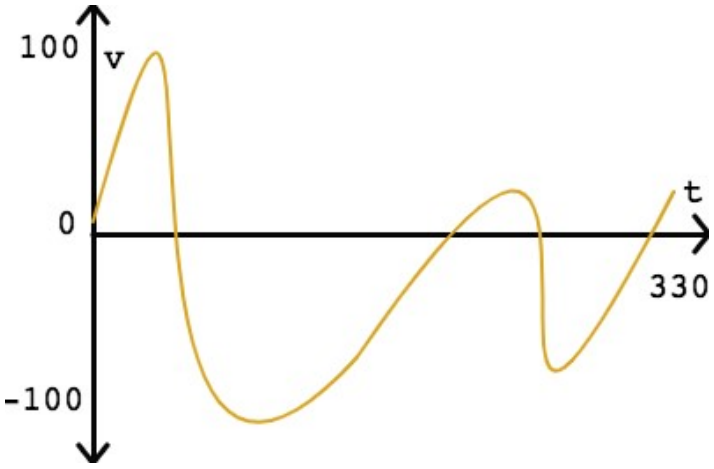
Now sing a little song to yourself. That's an infinite stream of audio information. Your voice is constantly changing in big and little ways, microsecond by microsecond.

Analog data is infinitely detailed. Computers can only store digital data, finite data in a binary representation.

So how can we capture the wondrous analog world of our senses and convert it into digital data? We can use a process of sampling, quantization, and binary encoding.

An analog signal

Let's start with a simple analog signal, a waveform representing a sound:



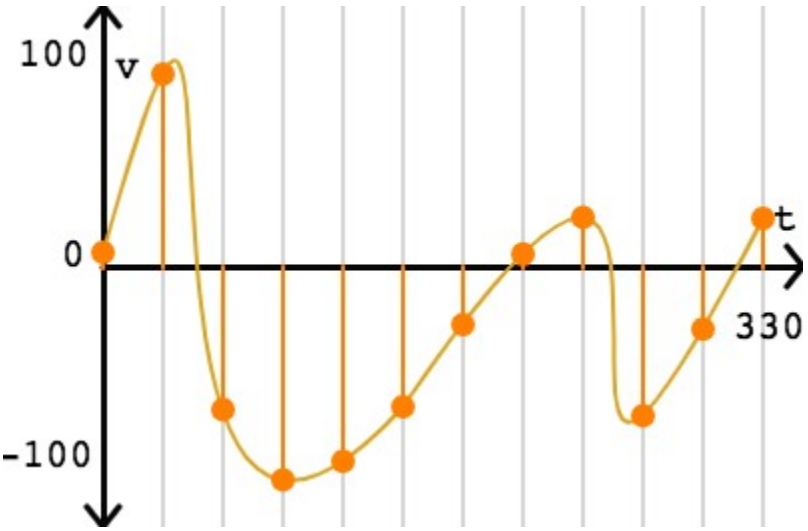
A graph with an x-axis labeled "t" that goes from 0 to 330 and a y-axis labeled "v" that goes from -100 to 100. A curvy line goes up and down across the graph.

All analog signals are continuous both in the time domain (x-axis) and in the amplitude domain (y-axis). That means that there is a precise value for every possible value of time, even as specific as "1.2345 seconds", and that value may be as precise as "47.8291824806423964 volts".

Sampling

The first step is **sampling**, where we take a sample at regular time intervals. This step reduces the continuous time domain into a series of discrete intervals.

In this signal, where time varies from 0 to 330 milliseconds, we could take a sample every 30 milliseconds:



A graph with an x-axis labeled "t" that goes from 0 to 330 and a y-axis labeled "v" that goes from -100 to 100. A curvy line goes up and down across the graph. A series of straight lines intercept the curvy line every 30 units on the x-axis.

That gives us 12 samples of the signal between 0 and 330 milliseconds.

Now we can express the signal as a series of sampled points:

(0, 7)

(30, 95.98676803710936)

(60, -71.43289186523432)

(90, -106.55949554687498)

(120, -97.21617085937501)

(150, -70)

(180, -29.045472375000003)

(210, 6.171340345703143)

(240, 24.439022283203116)

(270, -74.45763529492186)

(300, -31.31245312500002)

(330, 24)

The y-values are only as precise as our computer can store; numbers stored in computers aren't infinitely precise and may be rounded off.

The inverse of the sampling interval is the **sampling rate**: the number of samples in a second (or other unit of time). For example, a sampling interval of 30 milliseconds corresponds to a sampling rate of 33.33 samples per second (1/0.03s=33.333).

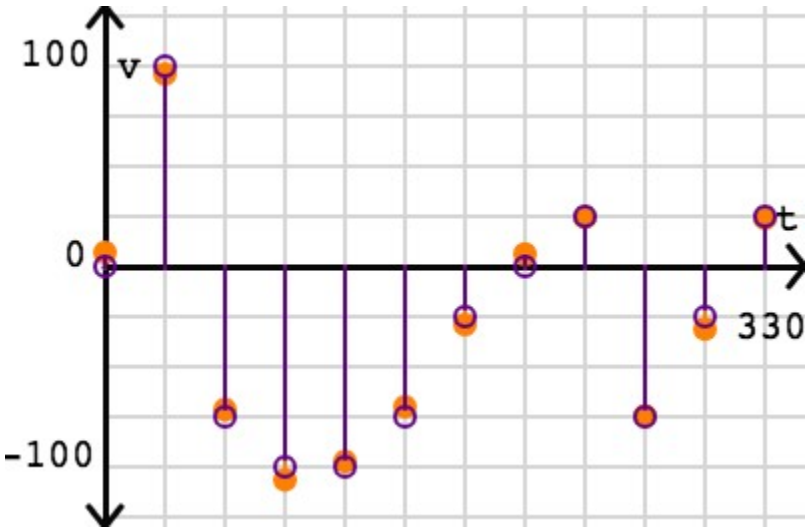
According to the Nyquist-Shannon sampling theorem, a sufficient sampling rate is anything larger than twice the highest frequency in the signal. The frequency is the number of cycles per second and measured in Hz (hertz). If a signal has a maximum frequency of 500 Hz, a sufficient sampling rate is anything greater than 1000 Hz.

A typical sampling rate for music recordings is 48 kHz (48,000 samples per second). That's a little over double the highest frequency that humans can hear, 20 kHz. If the audio only contains human speech, as is often the case for phone calls, a much smaller sampling rate of 8 kHz can be used since 4kHz is the highest frequency in most speech.

Quantization

After sampling, we are still left with a wide range in the amplitude domain, the y values. The next step of quantization reduces that continuous amplitude domain into discrete levels.

For our simple signal, where amplitude varies from -100 to 100 volts, we can apply a quantization interval of 25 volts:



A graph with an x-axis labeled "t" that goes from 0 to 330 and a y-axis labeled "v" that goes from -100 to 100. Sampled points are shown as orange circles. Lines go from the x-axis to near each of the sampled points, at an intersection with horizontal grid lines.

Now the 12 points all have y values that are multiples of 25:

(0, 0)

(30, 100)

(60, -75)

(90, -100)

(120, -100)

(150, -75)

(180, -25)

(210, 0)

(240, 25)

(270, -75)

(300, -25)

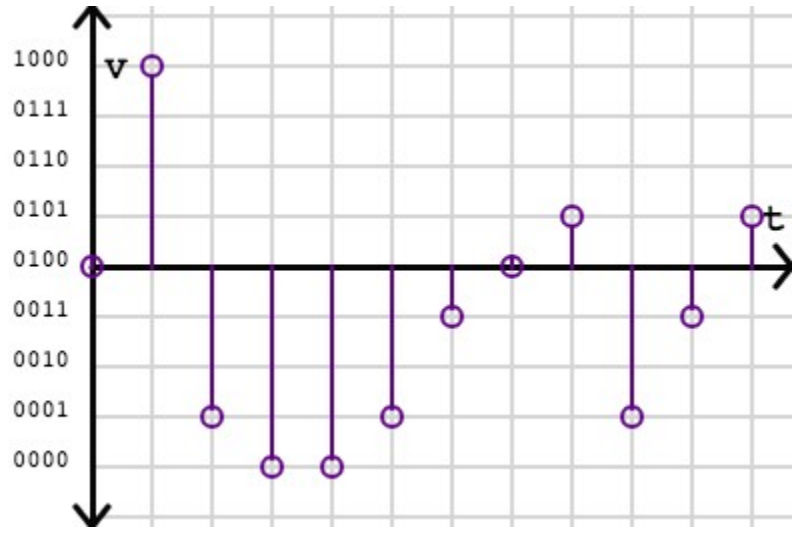
(330, 25)

The ideal quantization interval depends on our use case and physical constraints. If there is enough space to represent thousands of different y values, then we can use a very small quantization interval. If there is limited space, then we can use a large interval.

Binary encoding

That brings us to the final step: binary encoding. If there is a limited set of quantized y values, the computer does not need to store the actual value. Instead, it can store a much smaller value that represents the quantized y value.

For this signal, a quantization interval of 25 resulted in 9 possible y values. We can map the 9 values to the binary numbers 0000 - 1001:



A graph with an x-axis labeled "t" that goes from 0 to 330 milliseconds and a y-axis labeled "v" that goes from -100 to 100. A series of lines are shown every 30 milliseconds, with each line ending at a circle that intersects a horizontal grid line.

We can then encode the signal into this binary sequence:

```
0100 1000 0001 0000 0000 0001 0011 0100 0101 0001 0011 0101
```

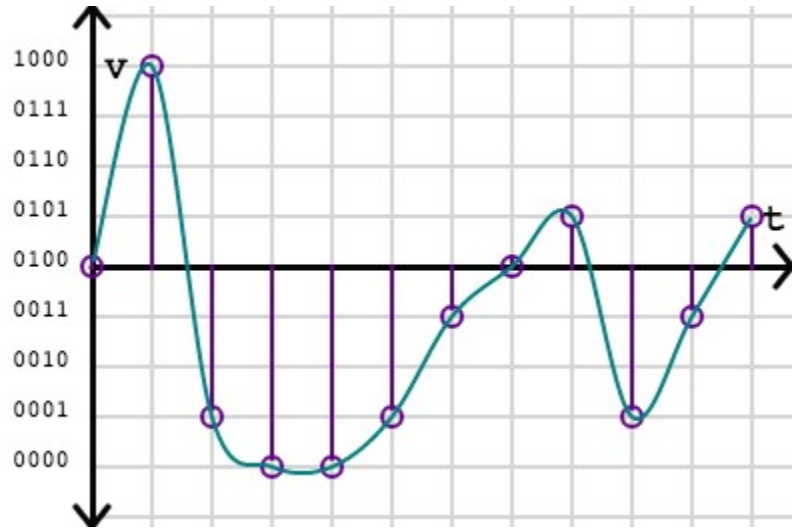
For a computer to understand that sequence, our digitized version would also need to include a description of how the sequence was sampled and encoded.

This encoding uses 4 bits per sample. The number of bits per sample is also know as the **bit depth**. The lowest bit depth is 1, which can only describe 2 values (0 or 1). The standard bit depth for telephone calls is 8 bits (256 values) and the recommended bit depth for YouTube music videos is 24 bits (over 16 million values).

Reconstruction

We often store analog signals in digital storage so that we can reproduce them later, like playing back an audio file or displaying an image. When a device wants to convert a digitized signal back into an analog signal, it will attempt to reconstruct the original continuous signal.

For this signal, a simple reconstruction strategy could interpolate a smooth curve through the quantized points:



A graph with an x-axis labeled "t" that goes from 0 to 330 milliseconds and a y-axis labeled "v" that goes from -100 to 100. A series of lines are shown every 30 milliseconds, with each line ending at a circle that

intersects a horizontal grid line. A curve is overlaid on top that joins those circles.

How well does that match the original? We can overlay the curves to see the difference visually:

A graph with an x-axis labeled "t" that goes from 0 to 330 milliseconds and a y-axis labeled "v" that goes from -100 to 100. A curvy line goes across the graph and is overlaid with another similar curvy line.

The reconstructed signal looks very close to the original but misses a few details. If we can decrease the sampling interval and lower the quantization error, we can bring the reconstructed curve closer to the original signal. We could also use different strategies for reconstructing the signal.

Summary

The first step of sampling converted an infinite stream to a finite sequence. In quantization, the values in that sequence were approximated. Finally, the values were encoded into bits for storage on a computing device. At some later point, a device could interpret those bits to attempt a reconstruction of the original infinite stream of continuous values.

Whenever we convert analog data to digital data, whether it's audio or visual, our goal is to sample data with enough precision so that we can

reconstruct it later at the desired quality level but not exceed our data storage capacity.

Land-line telephones use relatively low sampling rates and bit depths, since the data must travel over telephone lines, whereas movie directors record film at very high sampling rates and bit depths, so that they may replay it on giant screens later.