DTLTI Systems

Pieter P

Signals

A signal is a discrete function that maps an integer (the time step or point in time) to a real value or a vector of real values.

$$\begin{aligned} x &: \mathbb{Z} \to \mathbb{R} : & n &\mapsto x[n] \\ \vec{x} &: \mathbb{Z} \to \mathbb{R}^m : & n &\mapsto \vec{x}[n] \end{aligned} \tag{1}$$

In the simplest case, x will just map to a scalar ($\mathbb{R}$), but in general, it can also map to an m-dimensional vector ($\mathbb{R}^m$). This will be useful later, when we'll introduce systems with multiple inputs and outputs, or systems with multiple internal states.

A signal that is zero for all $n < 0$ is called a unilateral or one-sided signal.

Discrete-Time Linear Time-Invariant Systems

Discrete-Time Linear Time-Invariant Systems, or DTLTI systems for short, are systems that perform a linear transformation on signals. Time-invariance means that the transformation doesn't change over time, it is independent of the time step n, so it doesn't matter if you apply it to a certain signal now or in ten minutes, the resulting output signals will be the same.

A DTLTI system maps one signal to another:

$$T : (\mathbb{Z} \to \mathbb{R}^m) \to (\mathbb{Z} \to \mathbb{R}^m) : \vec{x}[n] \mapsto \vec{y}[n] \triangleq T(\vec{x}[n])$$

We'll define the properties of DTLTI systems mathematically:

> T is the transformation performed by a Discrete-Time Linear Time-Invariant (DTLTI) system if and only if
>
> 1. The transformation is linear:
>
> $T(a \cdot h[n] + b \cdot g[n]) = a \cdot T(h[n]) + b \cdot T(g[n])$
> 2. The transformation is time-invariant:
>
>   If $y[n] \triangleq T(h[n])$, then $\forall k \in \mathbb{N} : T(h[n-k]) = y[n-k]$

Nonlinear systems

As an example of a system that is not linear, consider the system $T : x[n] \mapsto \sqrt{x[n]}$. It is not linear, because
$T(2 \cdot 2) = \sqrt{4} = 2 \neq 2 \cdot T(2) = 2\sqrt{2}$.
Nonlinear systems are many times more complicated than linear systems, so they will not be covered in this series of articles.

Time-varying systems

The system $T : x[n] \mapsto n \cdot x[n]$ is not time-invariant, because the mapping explicitly depends on the time step n.
All systems in the following discussions will be time-invariant.

Example

Mathematical description

We'll define and plot a simple example signal, and then we'll apply a simple transformation to it. The input signal $x[n]$ is just a signal that oscillates between 3 and 1:

$$x : \mathbb{Z} \to \mathbb{R} : n \mapsto \cos(\pi n) + 2$$

The transformation maps each point of the signal to the average of the current value and the previous value:

$$T : (\mathbb{Z} \to \mathbb{R}) \to (\mathbb{Z} \to \mathbb{R}) : x[n] \mapsto \frac{x[n] + x[n-1]}{2}$$

The output signal $y[n]$ is defined as:

$$y[n] \triangleq T(x[n])$$

The input signal $x[n]$ and the output signal of the system $y[n]$ are plotted in the figure below.

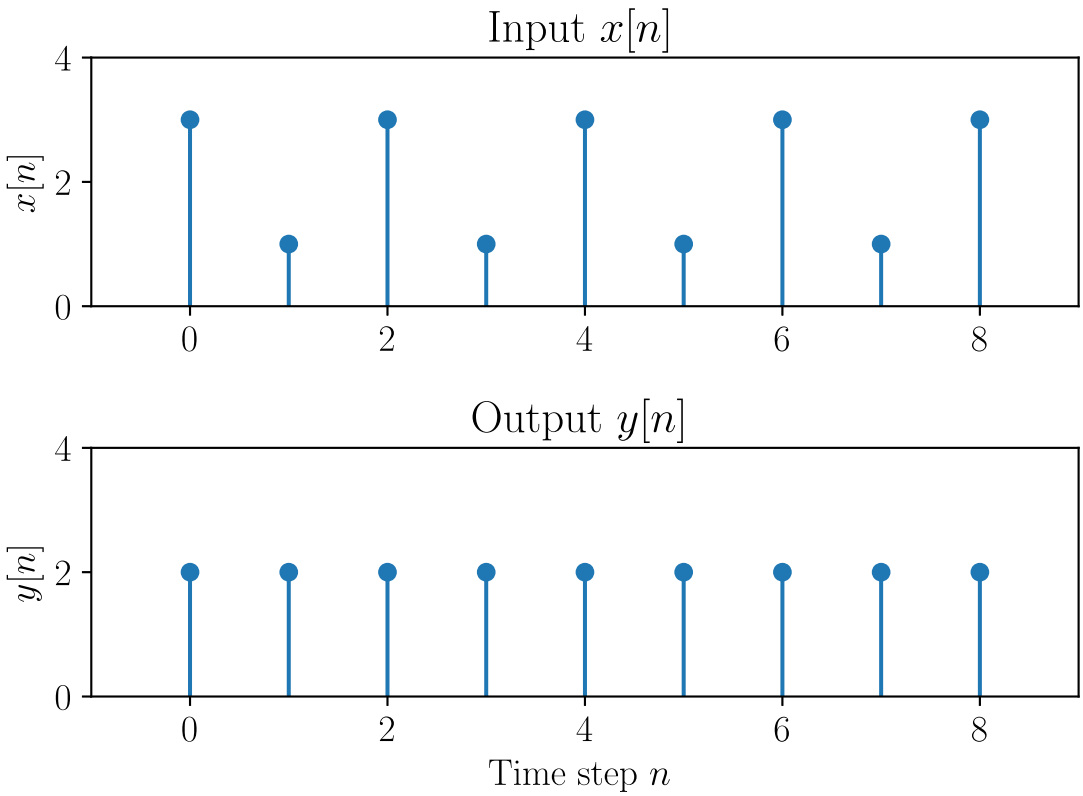

Image source code

Block diagram

The system could be implemented as follows.

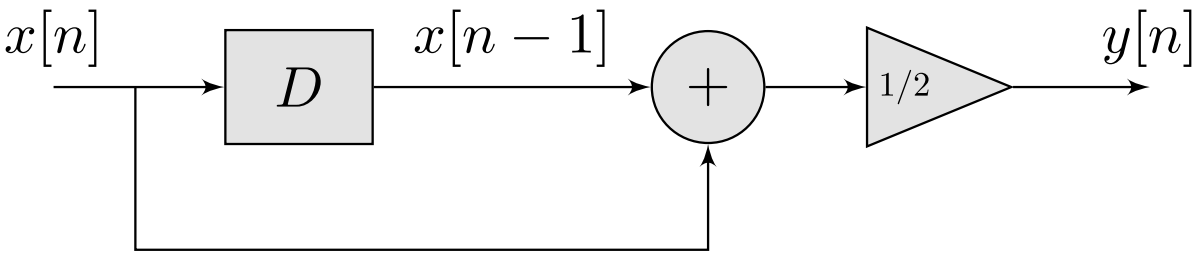

Image source code

The arrows indicate the direction of the data flow. The rectangular D block is called a delay or memory element, and it just delays the incoming signal by one time step. Sometimes, the Greek capital delta (Δ) is used instead, or in some contexts, it is indicated using z^{-1}, for reasons that will become apparent in the page on the Z-transform. The circle with the $+$ is a summator, it just adds all incoming signals together. Finally, the triangle containing a number is a scalar, and it just multiplies the signal with a constant factor.

Python implementation

A possible implementation of this system in Python is given in the code snippet below.
We just have to save the input to the delay element on each time step, because we need it to calculate the next output.

```
from numpy import array, arange, cos, pi

class ExampleDTLTISystem:
    def __init__(self, initial_state: float = 0.0):
        self.state = initial_state

    def __call__(self, x_n: float) -> float:
        # y[n] = (x[n] + x[n-1]) / 2
        y_n = (x_n + self.state) / 2.0
        # x[n] will be x[n-1] on the next time step,
        # so save it in the system's state
        self.state = x_n
        return y_n

n = arange(9)               # Create the time variable [0,1,2,…,7,8]
x = cos(pi * n) + 2         # Generate the input signal x[n]

T = ExampleDTLTISystem(1.0)  # Instantiate the system with x[-1] = 1
y = map(T, x)               # Apply the transformation y[n] = T(x[n])
```

It is a good exercise to try to understand how these three representations of the same system are related (the mathematical definition of T, the block diagram, and the Python implementation).