MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Spring Semester, 2008

Course notes for week 5, distributed March 4, 2008

Modeling and abstraction with signals and linear systems[1]

Note: This chapter contains questions througout labeled "self-check." These questions are to help you check your understanding as you are reading. They are *not* part of an assignment to be written up and turned in.

Our study of modeling and abstraction began with a look at functions, the PCAP framework and higher-order procedures for capturing common patterns. Then we considered issues of modularity, where we used the tools of object-oriented programming to model compound systems as collections of computational objects with encapsulated local state, leading to defining state machines as a common pattern of using objects for constructing models. In this chapter, we take up a different perspective on modeling, one that focusses not on the individual components in a compound system, but on the values that flow between components.

To illustrate this change in perspective, think about modeling a bank account that pays 0.2% a month, compounded monthly. At the beginning of every month, the account holder can make a deposit or withdrawal. Also, at the beginning of the month, the bank pays 0.2% interest based on the balance in the account. Using the methods of the previous chapter, we could model the bank account as a state machine, represented in the computer as an object whose local state is the account balance, where the state updates at monthly "steps." At each step the machine accepts the transaction amount (deposit or withdrawal) as an input and updates the balance. If we denote the local state by *balance* and let *input* (positive for deposits, negative for withdrawals) be the monthly transaction amount, then the state machine update rule at each step would be:

$$\mathit{balance} \leftarrow 1.002 \times \mathit{balance} + \mathit{input}$$

Figure 1 shows how we could picture the bank account as a state machine with an encapsulated state variable *balance*.

In this chapter, we'll view things differently. Rather than thinking about a single balance whose value changes at every step, we'll think in terms of the entire sequence of monthly balances. Similarly, rather than represent the monthly transaction (deposit or withdrawal) as a changing input value, we'll consider the entire sequence of transactions. More precisely, we'll let y be the sequence of balances, where $y[n]$ is the balance at month n, and x be the sequence of transactions where $x[n]$ is the amount deposited or withdrawn at month n. Then the bank account is modeled as:[2]

[1]These notes are based on earlier drafts by Leslie Kaelbling and the treatment heres draws extensively on slides and materials prepared for 6.003 by Denny Freeman.

[2]Alternatively, we could model the system as $y[n] = 1.002y[n-1] + x[n-1]$. That would be more consistent with our previous treatment of state machines, where the output at time n can depend on the input at time $n-1$, but not on the input at time n. For the signal processing perspective, it's natural to permit $y[n]$ to depend on $x[n]$, not just previous x's.

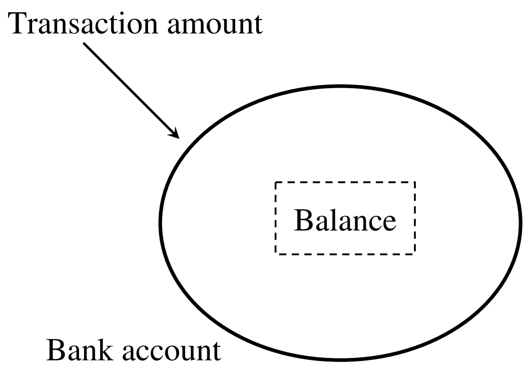


Figure 1: The bank account viewed as an object with the balance as an encapsulated variable.

$$y[n] = 1.002y[n-1] + x[n]$$

To be more concrete, suppose we start by depositing \$200, then on the next month deposit \$100, then withdraw \$100, then deposit \$100, and then another \$100, then two months with no transactions. The sequences x and y then begin:

n	$x[n]$	$y[n]$
0	200.00	200.00
1	100.00	300.40
2	−100.00	201.00
3	100.00	301.40
4	100.00	402.00
5	0.00	403.62
6	0.00	404.42
...	...	...

Figure 2 shows how we could visualize this, with the bank regarded as a *system* that transforms the sequence of transaction amounts x into the sequence of balances y. In contrast to figure 1 there's no state variable. Rather than individual variables whose values change, we focus on entire sequences of values and ask how these sequences are transformed. This approach is widely used in signal-processing applications, and the sequences are consequently referred to as *signals*.

State models and signal models are both important tools in the engineer's repertoire, and neither is better than the other in all circumstances.

State models tend to particularly appropriate in thinking about behavior changes in response to irregular stimuli. If you're designing an elevator, it's natural to use a state machine to describe how the elevator doors respond to button presses, press by press.

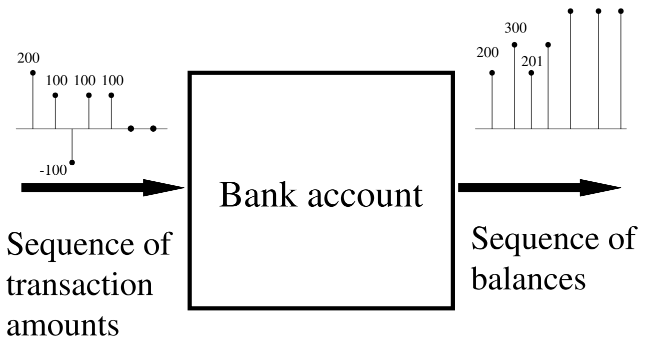


Figure 2: The bank account viewed as a system that transforms a sequence of transactions to a sequence of balances.

Signal models tend to be valuable when we want to understand patterns of behavior over time. To illustrate what we mean by patterns of behavior over time, suppose you're programming a robot to drive down the center of a narrow corridor. At regular intervals, your program reads the robot sensors, estimate the distances to the walls, and adjusts the robot's right and left wheel velocities in an attempt to make it stay on track. If all goes well, the robot will remain close to the center, making small corrections to stay on track. But if you're unlucky, the robot might begin to overcompensate in its turning, making larger and larger turns and eventually crashing into the wall. We can view the robot program as a signal-processing system, where the input signal is how far the robot has deviated from the center of the corridor, and the output signal is the robot's turning velocity. One of the things we might want to know about the system is whether it is *stable*, i.e., whether the robot stays on track and corrects for small deviations; as opposed to *unstable*, i.e., whether small deviations grow and grow. As we'll see, the theory of signal processing will give us tools to analyze systems like the robot program to determine whether they are stable, and provide design techniques we can use to avoid instability.

Signals and signal operations

We'll start our study of signal-processing models by looking at signals and the means for combining them. We'll define a *signal* to be a function that takes an integer argument and returns a numeric value. If x is a signal, we write $x[n]$ to be the value of x at n. Typically, we think of n as measuring time, like the nth month for the bank account, or the nth time the robot control loop runs. We'll regard signals in principle as stretching into the indefinite future and the indefinite past (i.e., n can be arbitrarily large in the positive or negative direction). In practice, for the examples in this course, we'll assume there is some starting value (typically $n = 0$) before which all the signal values are 0.[3]

Given two signals x_1 and x_2, their *sum* is the signal whose value at n is $x_1[n] + x_2[n]$. In symbols:

$$(x_1 + x_2)[n] = x_1[n] + x_2[n]$$

We can also *scale* a signal by a constant: if c is a number and x is a signal, then cx is the signal whose value at n is $c \times x[n]$:

$$(cx)[n] = cx[n]$$

Adding and scaling satisfy the familiar algebraic properties of addition and multiplication, namely, addition and scaling are commutative and associative, and addition distributes over scaling. That is, as functions we have

$$c(x_1 + x_2) = (cx_1 + cx_2)$$

which we can verify by checking that they have the same value for any n:

$$\begin{aligned} c(x_1 + x_2)[n] &= c \cdot (x_1 + x_2)[n] \\ &= c \cdot (x_1[n] + x_2[n]) \\ &= cx_1[n] + cx_2[n] \\ &= (cx_1)[n] + (cx_2)[n] \\ &= (cx_1 + cx_2)[n] \end{aligned}$$

Going along with these operations is a traditional way of denoting them by means of *block diagrams*, shown in figure 3. Addition of signals is represented by an *adder* block with two signals entering and the sum leaving; scaling is indicated by a triangle (deriving from the electrical symbol for an amplifier) with the scale factor written on it.

There's one more operation on signals: *delay.* The delay of a signal x is the signal whose value at n is $x[n-1]$. That is, whatever x does, the delayed signal does it one unit later: if $x[3] = 10$ then the delayed signal is 10 when $n = 4$. We denote the delayed signal by $\mathcal{R}x$, so that for any n we have:

$$(\mathcal{R}x)[n] = x[n-1]$$

The use of the letter R comes from "right shift": in pictures, the delayed signal is just the original signal shifted one unit to the right. In block diagrams, we represent delay as a box with the word "delay" written on it, which inputs x and outputs Rx as shown in figure 4. Sometimes we write "R" on the box, rather than "delay."

Applying the delay operator to a signal x isn't actually multiplying the signal by a quantity $\mathcal{R}$, but in terms of symbols, the operation behaves that way, following simple algebraic laws with respect to multiplication and scaling: The delay of the sum of two signals is the sum of the delays, and the delay of a scalar multiple is the scalar multiple of the delay:

$$\begin{aligned} \mathcal{R}(x_1 + x_2) &= \mathcal{R}x_1 + Rx_2 \\ \mathcal{R}(kx) &= k \cdot \mathcal{R}x \end{aligned}$$

[3]There are important application areas for signal processing where this is not the best assumption. For example, if we're doing media processing, it's generally more convenient to regard audiovisual signals as originating in the indeterminate past, rather than as starting at a particular instant.

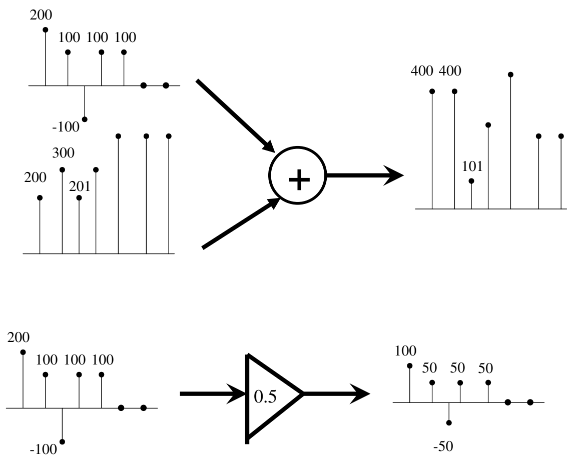


Figure 3: Block diagram symbols for adding and scaling signals.

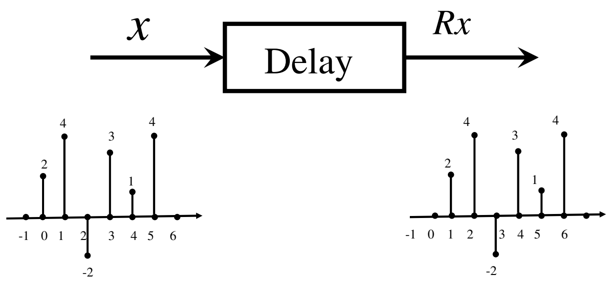


Figure 4: The delay operator $\mathcal{R}$ shifts a signal one unit to the right.

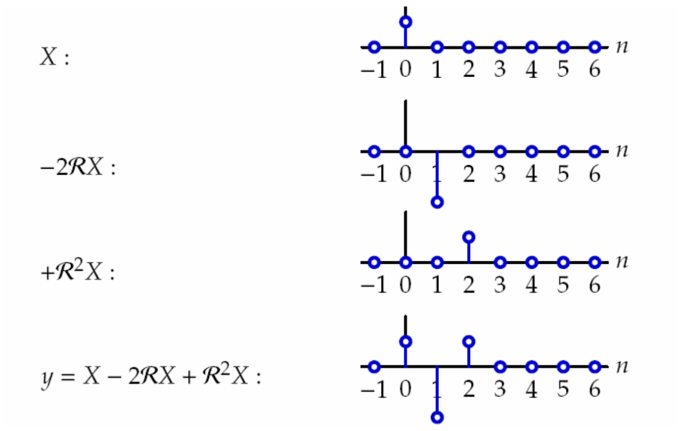


Figure 5: Generating a new signal from an original input by combining adding, gain, and delay.

As a result, we can manipulate expressions with signals, just like algebra: in terms of our PCAP framework, the *means of combination* that govern addition, scaling, and delay of signals are just the familiar rules of algebra. Figure 5 shows an example of beginning with a signal x and subtracting 2 times a delayed version of x and then adding a double delayed version of x. If we denote the resulting signal by y then for any n we have

$$y[n] = x[n] - 2x[n-1] + 2x[n-2]$$

The original signal x in figure 5, which is 1 at $n = 0$ and 0 otherwise, has a special name: It is called the *unit sample* and denoted by δ. It is defined by the property:

$$\delta[n] = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases}$$

Self-check 1: Suppose x is the signal with $x[0] = 10$, $x[2] = 8$, and $x[6] = 1$. Write x as a sum of scaled and delayed copies of the unit sample.

Self-check 2: Show that any signal x with $x[n] = 0$ for $n < 0$ is a sum of scaled and delayed copies of the unit sample.

Linear time-invariant systems

By a *system* we mean something that transforms an input signal to an output signal. The bank account system of figure 2 transforms the input signal of monthly transaction amounts into the output signal of monthly balances.

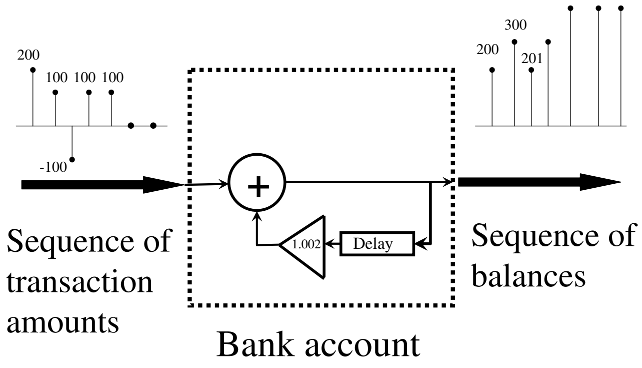
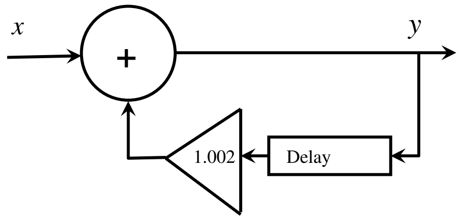


Figure 6: The bank account system of figure 2, implemented as a delay, adder, and gain.

$y[n] = 1.002y[n-1] + x[n]$



$y = 1.002\mathcal{R}y + x$

Figure 7: The bank account in three different representations: a difference equation; a delay-adder-gain block diagram; and an operator equation.

One way to create systems is as combinations of addition, scaling, and delay. Figure 6 shows how to build the box labeled "bank account" in figure 2 by combining an adder, a gain element, and a delay.

Figure 7 shows the same three elements, this time without the enclosing "bank account" box, as a system that transforms an input signal x to an output signal y. As the figure indicates, we can represent the system in three different ways:

1. As a *difference equation.* Here the equation is $y[n] = 1.002y[n-1] + x[n]$. In general, it's common to denote the input signal by x and the output signal by y.

2. As a *delay-adder-gain block diagram*, by which we mean a block diagram created by wiring together delay, adder, and gain elements (and no other kinds of elements) to create a system that transforms an input signal to an output signal.

3. As an *operator equation* in terms of the signals x and y and the delay operator $\mathcal{R}$. Here the equation is $y = 1.002\mathcal{R}y + x$

When we restrict to block diagrams with only delays, adders, and gains, we're limited to a tiny subset of the class of all possible systems. This is called the class of *linear time-invariant* systems or *LTI* systems. In terms of equations, the LTI limitation means that the difference equations are linear in the $x[k]$ and $y[j]$ (i.e., no higher powers, or non-linear functions like sines or cosines, of these terms) and that the terms have constant coefficients. The general equation has the form

$$a_0y[n] + a_1y[n-1] + \cdots + a_ky[n-k] = b_0x[n] + b_1x[n-1] + \cdots + b_jx[n-j]$$

That is, the x's and y's can go back different depths in history.[4] Observe that, because the coefficients are constant and thus do not vary with n, we can shift the equation forward or backward in n without changing the system. For example, the equations

$$3y[n] + 4y[n-1] - 7y[n-2] = 10x[n] + 6x[n-2] + 8x[n-3]$$

and

$$3y[n-2] + 4y[n-3] - 7y[n-4] = 10x[n-2] + 6x[n-4] + 8x[n-5]$$

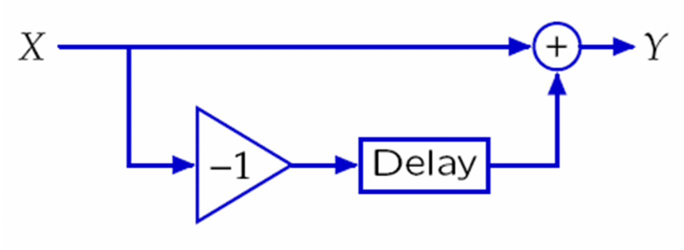
represent the same system—it's just a matter of substituting $n-2$ for n on both sides of the equation.

Equations restricted to this form are called *linear difference equations with constant coefficients* and the corresponding systems are called *linear time-invariant systems* or *LTI systems.* They are also called *linear shift-invariant* (LSI) systems, which is especially appropriate in applications like image processing, where the signals are not necessarily time signals. Restricting our study to LTI systems might seem severe—and it is—but in return we gain access to a powerful analytical theory of LTI systems that is central to engineering design.

A good way to start thinking about LTI systems is to practice translating among the three representations. If we're given a digram and asked to produce the equation, we can sometimes do this by inspection, by writing down the output of the adder. Figure 8 shows two examples. In the top

[4]If we have a system where the x's and y's don't start with the same n we can put it in this form by taking various a's and b's to be 0. For example, we would write the equation $y[n] + 2y[n-1] = x[n-2]$ as $y[n] + 2y[n-1] = 0 \cdot x[n] + 0 \cdot x[n-1] + x[n-2]$.

$y[n] = x[n] - x[n-1]$ $y = x - \mathcal{R}x$



$y[n] = x[n] - 2x[n-1] + x[n-2]$ $y = x - 2\mathcal{R}x + \mathcal{R}^2x$

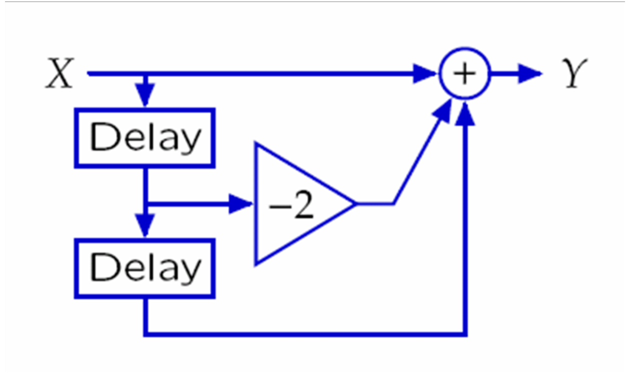


Figure 8: Example LTI systems, showing the three representations. The bottom system has a double delay

system the output signal y is the sum of the input x plus the delay of the negative of x (which is the same as the negative of the delay of x), the the equation is $y = x - \mathcal{R}x$, or $y[n] = x[n] - x[n-1]$. For the bottom, y is the sum of three terms: x, negative two times the delay of x, and x delayed twice: $y = x - 2\mathcal{R}x + \mathcal{R}^2x$, or $y[n] = x[n] - 2x[n-1] + x[n-2]$.

Notice that it's straightforward to go between the difference equation and the operator equation: just replace each $x[n-k]$ by $\mathcal{R}^kx$, and similarly for y. Going between the equations and the diagrams is not so straightforward. In general, there are many delay-adder-gain diagrams that can lead to a given equation. Picking one is an exercise in design.

Things are not so simple if there is more than one adder, as in figure 9, which not only has multiple adders, but a feedback path from the output of the delay back to left-most adder.

A good way to derive the equation is to write down a separate equation for each adder, naming the outputs of the adders if necessary, and then manipulating the equations by algebra to end up with a single equation that relates the input x to the output y. To apply this in figure 9 we name the output of the left-hand adder w. Then at the left-hand adder we have the equation:

$$w = x + \mathcal{R}w$$

and at the right-hand adder we have

$$y = w - \frac{1}{2}\mathcal{R}w$$

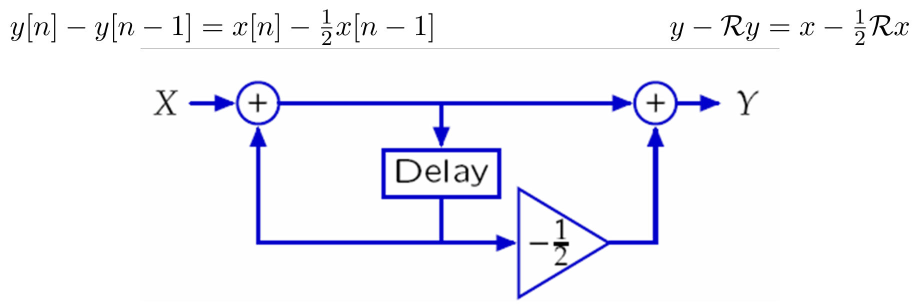

$y[n] - y[n-1] = x[n] - \frac{1}{2}x[n-1]$
$y - \mathcal{R}y = x - \frac{1}{2}\mathcal{R}x$
X
+
Delay
$-\frac{1}{2}$
+
Y

Figure 9: An LTI system, with multiple adders and feedback. We can derive the equation that relates x and y by naming the output of the left-hand adder w and combining equations.

In order to combine these equations we use ordinary algebra to solve the first equation for w in terms of x and substitute into the second equation. In our algebra, we act as if $\mathcal{R}$ were an ordinary variable. So, for the first equation we have:

$$\begin{aligned} w &= x + \mathcal{R}w \\ x &= w - \mathcal{R}w = w(1 - \mathcal{R}) \\ w &= \frac{x}{1 - \mathcal{R}} \end{aligned}$$

and substituting into the second equation gives:

$$\begin{aligned} y &= w - \frac{1}{2}\mathcal{R}w \\ &= \frac{x}{1-\mathcal{R}} - \frac{1}{2}\mathcal{R}\frac{x}{1-\mathcal{R}} \\ y(1-\mathcal{R}) &= x - \frac{1}{2}\mathcal{R}x \\ y - \mathcal{R}y &= x - \frac{1}{2}\mathcal{R}x \end{aligned}$$

or, if we prefer difference equations:

$$y[n] - y[n-1] = x[n] - \frac{1}{2}x[n-1]$$

Self-check 3: Derive the operator equation and the difference equation for each of the LTI systems shown in figure 10.

Self-check 4: For each of the following difference equations, transform it into an equivalent operator equation and sketch an adder-gain-delay block diagram for a system that realizes the equation.

(a) $y[n] = x[n] - 2x[n-1] + 3x[n-2]$

(b) $y[n] + y[n-1] = x[n] + 2x[n-1]$

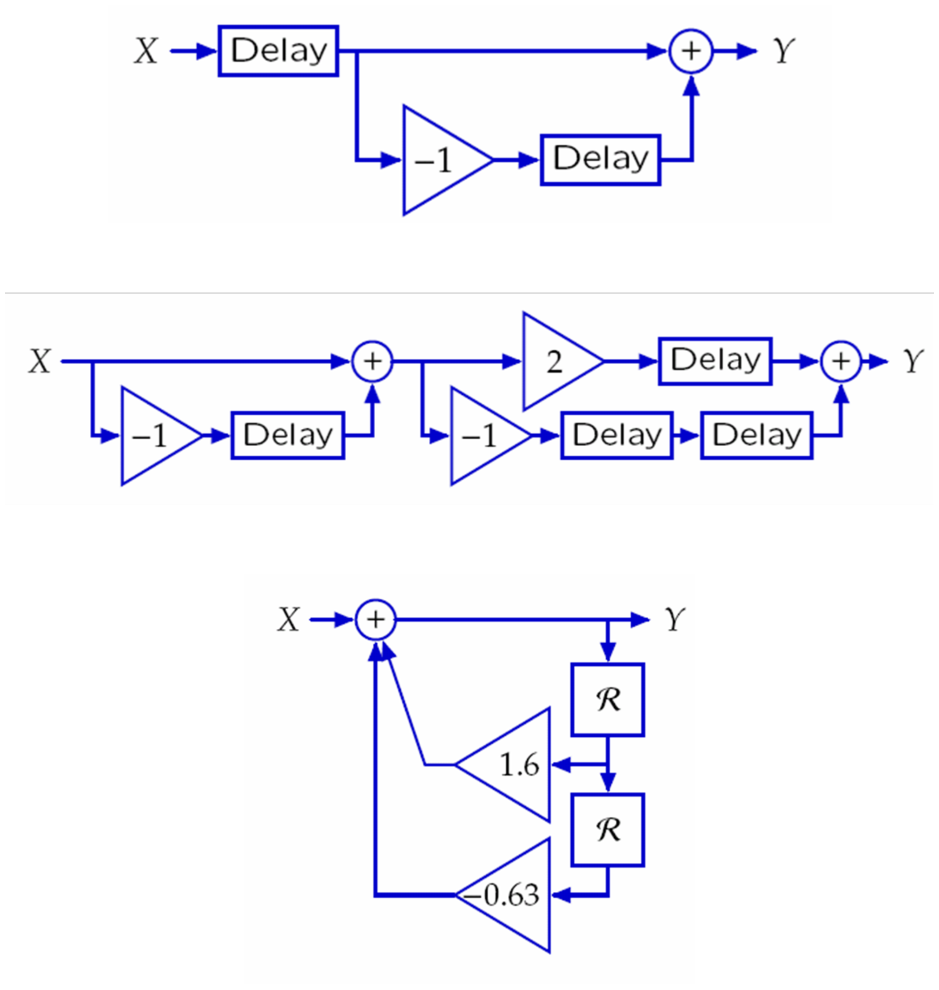


Figure 10: Derive the operator equation and the difference equation for each of these LTI systems

System functions

There's something suspicious about doing algebraic manipulations with operator equations. It seems fine to add signals and multiply them by constants, even to represent delay as multiplication by $\mathcal{R}$. But how should we interpret an expression like

$$w = \frac{x}{1 - \mathcal{R}}$$

The answer is that there is a mathematical trick that we can use to interpret any "fractions" in terms of multiplication. For example, imagine a world where we knew only about integers, and we want to interpret the equation $w = 2/5$. One way to do that would be to say that w is the thing that when you multiply it by 5 gives 2. That's a perfectly good definition.

Similarly, with signals, we can say that $w = x/(1 - \mathcal{R})$ means that when you multiply w by $1 - \mathcal{R}$, i.e., subtract $\mathcal{R}w$ from w, you get x. In other words, we can interpret division in operator arithmetic by saying that result of a division is the thing that makes the equalities hold when the you multiply through to clear the denominators.[5]

In general, if we have an LTI system described by an equation relating the output signal y to the input signal x, we define the *system function* to be "y divided by x," i.e., the expression that you multiply x by to get y. For example, we found that the two systems shown above in figure 8 are described by the equations:

$$y = x - \mathcal{R}x$$

and

$$y = x - 2\mathcal{R}x + \mathcal{R}^2 x$$

The system function for the first is

$$H = \frac{y}{x} = \frac{x - \mathcal{R}x}{x} = 1 - \mathcal{R}$$

For the second, the system function is

$$H = \frac{y}{x} = \frac{x - 2\mathcal{R}x + \mathcal{R}^2 x}{x} = 1 - 2\mathcal{R} + \mathcal{R}^2$$

For the system in figure 9, we found the signal equation to be

$$y - \mathcal{R}y = x - \frac{1}{2}\mathcal{R}x$$

and so we can compute the system function as

$$\begin{aligned} y - \mathcal{R}y &= x - \frac{1}{2}\mathcal{R}x \\ y\,(1 - \mathcal{R}) &= x\left(1 - \frac{1}{2}\mathcal{R}\right) \\ H &= \frac{1 - \frac{1}{2}\mathcal{R}}{1 - \mathcal{R}} \end{aligned}$$

[5]For later rivisions: Add a footnote here to say when, in general, one can do this mathematical trick (division algebras). Also mention that in general, there are conditions on the signals that must be satisfied for the operator algebra to fully mirror the sytem behavior.

For the system described by the difference equation

$$a_0y[n] + a_1y[n-1] + a_2y[n-2] = b_0x[n] + b_1x[n-1] + b_2x[n-2]$$

the operator equation is

$$a_0y + a_1\mathcal{R}y + a_2R^2y = b_0x + b_1\mathcal{R}x + b_2\mathcal{R}^2x$$

and the system function is

$$H = \frac{y}{x} = \frac{b_0 + b_1\mathcal{R} + b_2\mathcal{R}^2}{a_0 + a_1\mathcal{R} + a_2\mathcal{R}^2}$$

Notice that the system function has the form of the ratio of two polynomials in $\mathcal{R}$.

By the way, to be precise, we should admit that the system function H isn't exactly a function as we've defined it. Rather it's an expression in the algebra of operators.

For the general LIT system defined by an equation of the form

$$a_0y[n] + a_1y[n-1] + \cdots + a_ky[n-k] = b_0x[n] + b_1x[n-1] + \cdots + b_jx[n-j]$$

The system function is

$$H = \frac{b_0 + b_1\mathcal{R} + \cdots + b_j\mathcal{R}^j}{a_0 + a_1\mathcal{R} + \cdots + a_k\mathcal{R}^k}$$

Given a system function, we can recover the difference equation by multiplying by the denominator to get an operator equation and converting the result to difference equation form. For example, if the system function is

$$H = \frac{y}{x} = \frac{5 + 7R - 8R^2}{3 - R^2 + 6R^4}$$

we multiply through to get the operator equation

$$3y - R^2y + 6R^4y = 5x + 7Rx - 8R^2x$$

which gives the difference equation

$$3y[n] - y[n-2] + 6y[n-4] = 5x[n] + 7x[n-1] - 8x[n-2]$$

Once we have the difference equation, we can start with any input sequence $x[n]$ and some intitial conditions and use a computer program to generate the resulting sequence $y[n]$ io the form

$$y[n] = \frac{1}{3}\left(y[n-2] - 6y[n-4] + 5x[n] + 7x[n-1] - 8x[n-2]\right)$$

That's useful, but our goal in this chapter and the next is to learn about the general qualitative behavior of systems, not just particular numerical outputs generated from particular numerical inputs. As we'll see, the system function lets us get at this qualitative behavior directly.

Self-check 5: Find the system function for each of the systems in figure 10.

Self-check 6: Suppose that sequences w, x, and y are related by the equations:

$$\begin{aligned} w[n] - w[n-1] &= x[n] - 2x[n-1] \\ y[n] - y[n-1] &= w[n] - w[n-2] \end{aligned}$$

Find a difference equation that relates y directly to x. Hint: Use system functions and algebra.

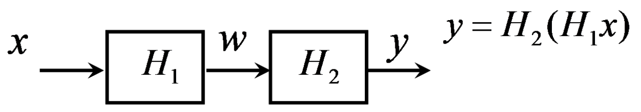


Figure 11: The system function of a cascade is the product of the two system functions.

System functions as means of abstraction

We've defined the system function to be "the thing you multiply the input x by to get the output y," which immediately gives us the operator equation:

$$y = Hx$$

Or said another way: *If you start with the input, and multiply by the system function, you get the output.* For this reason, the system function is also sometimes referred to as the *transfer function*, because it describes the relation between the input and output of the system. It's common, by an abuse of language, to use H to denote the system itself, thus speaking of "the system H" when we actually mean "the system whose system function is H."

This might all seem like trivial rewording of definitions, but there's something profound here. What's profound is that the behavior of linear time-invariant systems can be described by algebraic manipulations built from three primitive operations: addition, scaling, and delay—and where the means of combination—algebra addition and multiplication—obey the familiar commutative, associative, and distributive laws.

This simple statement has major consequences. For instance suppose we have two systems, with system functions H_1 and H_2, and we cascade them by feeding the output of H_1 into the input of H_2 as shown in figure 11. . Letting x denote the input to H_1 and w denote the output of H_1, we have

$$w = H_1x$$

Feeding w into H_2 and letting y denote the output, we have

$$y = H_2w$$

Thus for the overall cascade of H_2 after H_1 we have

$$y = H_2w = H_2(H_1x) = (H_2H_1)x$$

Thus

> *The system function of a cascade of two systems H_2 after H_1 is the product of the system functions H_2H_1.*

Now suppose we had arranged the cascade in the other order, with x feeding into H_2 to get an output v and v feeding into H_1 to get y, as shown in the bottom of figure 12. Then we'd have

$$y = H_1v = H_1(H_2x) = (H_1H_2)x$$

so the system function would be H_1H_2. But H_1H_2 is equal to H_2H_1 because multiplication commutes: the two cascades have the same system function. Since the system function describes the input-output behavior of the system, we have therefore shown:

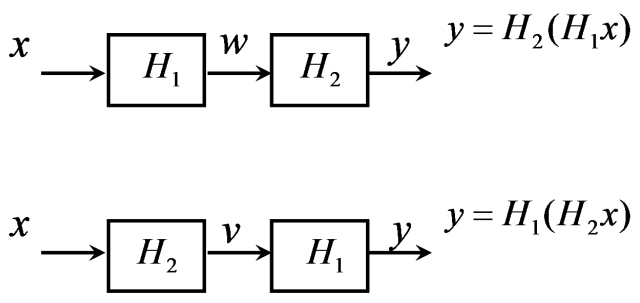


Figure 12: For the system function—and hence the input-output behavior—the order of systems in a cascade doesn't matter.

> *For the cascade of two linear time-invariant systems, the output doesn't depend on the order of the cascade.*

This may seem obvious if we think in terms of multiplication of system functions. But it's far from obvious in terms of block diagrams. Figure 13 shows an example of starting with a cascade of two systems at the top of the digram and deriving the equivalent form at the bottom by interchanging the order of the cascade. If you try working out the two system functions independently, you'll see that the computations involved are very different and the fact that they produce the same answer might seem remarkable.

The fact that the input-output behavior of a cascade does not depend on the order for the cascade is a very special property of linear time-invariant systems. It's not true even for the simplest systems that do not obey linearity and time-invariance. This is also much more than an abstract mathematical statement. It means that when we design real signal processing systems as cascades of components, we can arrange the cascades in any order.[6]

Self-check 7: Consider the simple LTI system $y[n] = 2x[n]$ and also the simple *non-linear* system given by $y[n] = x[n]^2$. Show that the order of cascade *does* matter in combining these two systems.

Self-check 8: Find the system function of the system in figure 13.

Here's another easy consequence of our algebraic view of systems: Suppose we combine two systems H_1 and H_2 in parallel as shown in figure 14. If the two component systems are given by:

$$y_1 = H_1x$$

and

$$y_2 = H_2x$$

[6]As with all things that seem too good to be true, there's a tricky "gotcha" to beware of. Looking ahead a to our study of circuits, it's *not* the case that if you simply combine two circuits by wiring the output of one to the input of the next, you're not cascading them as LTI systems, because of loading effects, unless you *isolate* the two circuits using an op-amp.

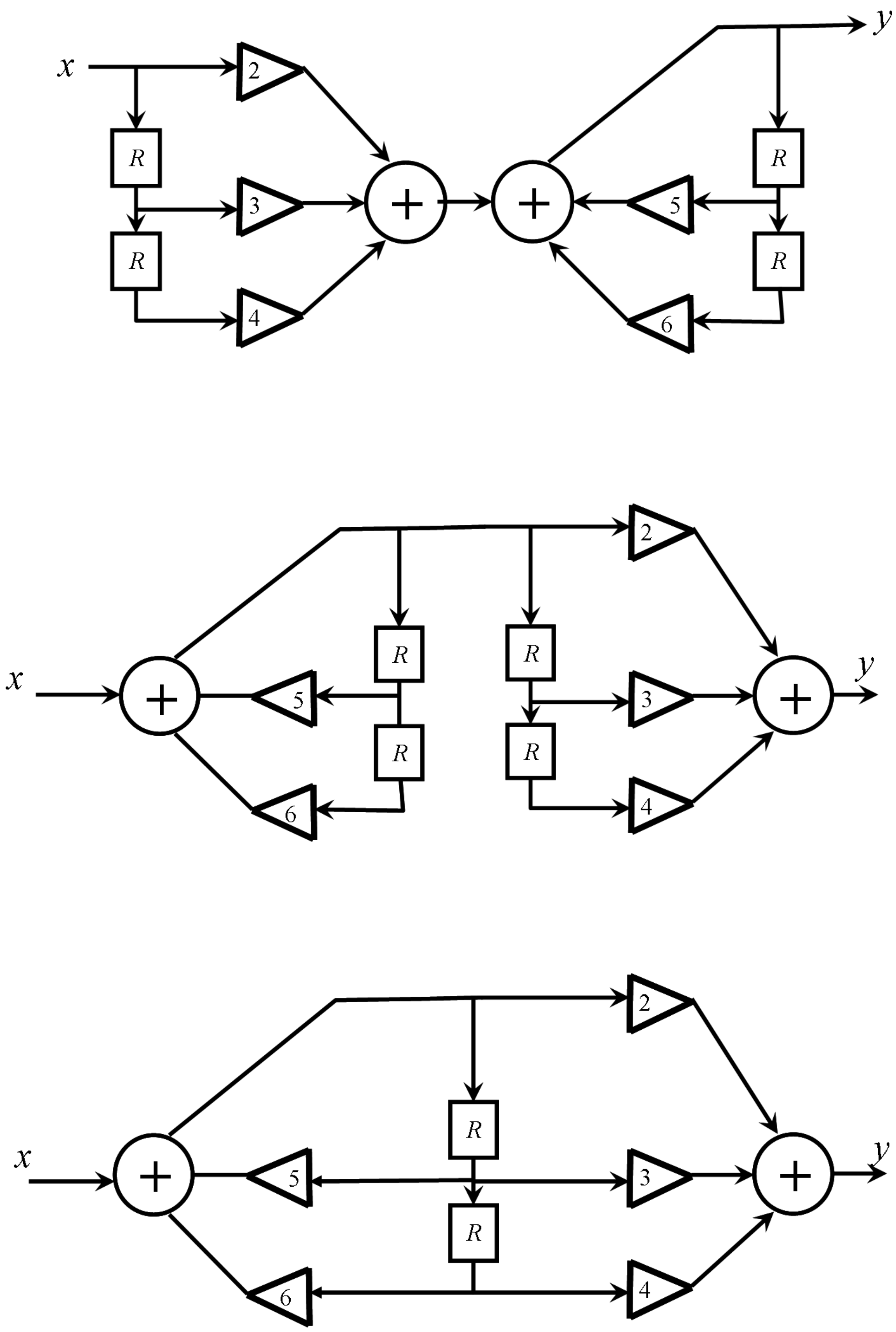


Figure 13: Interchanging the order of a cascade: The three systems are all equivalent. (Adapted from Wm. Siebert, *Circuits, Signal, and Systems*.)

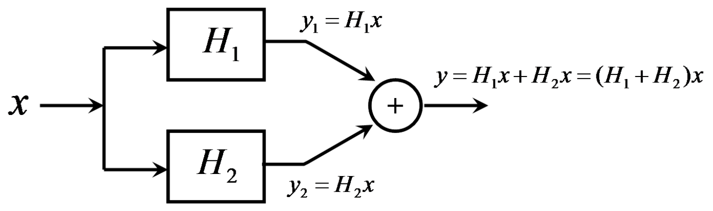


Figure 14: The system function of a parallel combination is the sum of the system functions of the components.

then for the sum, we have

$$y = y_1 + y_2 = H_1 x + H_2 x = (H_1 + H_2)x$$

and so for the system function we have

$$H = \frac{y}{x} = H_1 + H_2$$

That is,

The system function of a parallel combination is the sum of the system functions of the components.

Another important fact about system functions is one that we've already been using, but it bears calling out explicitly:

If two systems have the same system function then they have the same input-output behavior, i.e., for any given input, the two systems have the same output.

Again, this seems like a tautology, given how we've defined the system function. But the implications are far from obvious in terms of block diagrams. Consider the top system in figure 15. We can compute the system function by considering the output of the adder and doing algebra:

$$\begin{aligned} y &= x + 5\mathcal{R}y - 6\mathcal{R}^2 y \\ x &= y\left[1 - 5\mathcal{R} + 6\mathcal{R}^2\right] \\ H &= \frac{y}{x} = \frac{1}{1 - 5\mathcal{R} + 6\mathcal{R}^2} \end{aligned}$$

But also by algebra, we can factor the denominator to get

$$H = \frac{1}{1 - 5\mathcal{R} + 6\mathcal{R}^2} = \frac{1}{1 - 2\mathcal{R}} \cdot \frac{1}{1 - 3\mathcal{R}}$$

which means that we can express the *same system* as a cascade of a system with $H_1 = 1/(1 - 2\mathcal{R})$ and a system with $H_2 = 1/(1 - 3\mathcal{R})$, where the cascade can be in either order. In other words, the two delay-adder-gain block diagrams in figure 15 have the same system function and hence all the same input-output behavior, something that is hardly apparent from looking at the diagrams.

$$H = \frac{y}{x} = \frac{1}{1-5R+6R^2} = \frac{1}{1-2R}\cdot\frac{1}{1-3R}$$

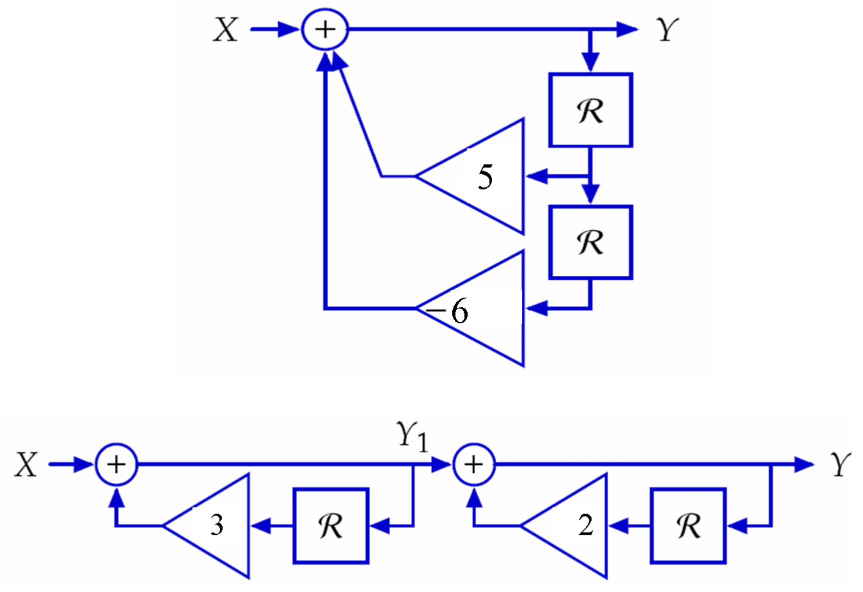


Figure 15: Factoring the denominator shows that both systems have the same input-output behavior.

As we'll see in the next chapter, this technique of "factoring" system functions by factoring the denominator will be central to our analysis of LTI systems.

Before going on, let's summarize where we've gotten to, using the PCAP framework applied to linear systems:

- We can construct linear systems out of three *primitive elements*: delays, adders, and gains.
- For *means of combination* we can combine these elements into systems by wiring them together.
- Remarkably, we can *abstract* a system as a system function: As far as input-output is concerned, all those delays, adders, and gains, can be summed up for a system in a *single* ratio of polynomials.

And there's more: Now that we have systems and system functions, we can combine and abstract *them*:

- The *primitives* are the systems, represented by their system functions.
- The *means of combination* are cascading and parallel combination, which correspond to multiplication and addition when viewed in terms the system function.

- The *means of abstraction* come from the fact that we can consider the algebraic combination of system functions to be just another system function. At the end of the day, any system function that we've built up is nevertheless still a ratio of polynomials, which we can regard as the system function for the composite system.

In other words, LTI systems have the special property that the means of abstraction (system function) turns manipulating LTI systems into ordinary algebra, and so all the techniques of algebra are available for the analysis of LTI systems. We'll draw heavily on this idea in the next chapter, where we'll also see examples of the PCAP framework's common patterns use for LTI systems—in particular, feedback.

Qualitative Behavior of LTI systems: Preview

We began this chapter by introducing signal models as alternatives to state models, and said this would be valuable in gaining insight into qualitative patterns of system behavior over time. We've introduced some ideas—LTI systems, difference equations, block diagrams, system functions—and described their properties and shown how to manipulate them. But we haven't actually *used* any this machinery to gain insight into any system behavior. Actually, we hardly looked at system behaviors at all. That's what we'll do in the next chapter. First, however, we'll give a quick introduction to show you where this story is going.

Let's return to where we began the chapter: a bank account that pays compound interest at a rate of $r\%$ per period. If we write $p = 1 + r$, then the system is described by the difference equation relating the monthly balance y to the monthly transaction amount x

$$y[n] = py[n-1] + x[n]$$

or, as we now know, by the system function

$$H = \frac{y}{x} = \frac{1}{1 - p\mathcal{R}}$$

or by the delay-adder-gain block diagram in figure 16.

Suppose we start with a one dollar and never make any more deposits or withdrawals. Then the difference equation reduces to:

$$y[n] = \begin{cases} 1 & \text{if } n = 0 \\ py[n-1] & \text{if } n > 0 \end{cases}$$

In other words, the output $y[n]$ is an exponential sequence with exponential factor p.

Figure 16 plots this sequence for four values of p: 1.5, 0.8, -1.5, and -0.8. Let's examine the qualitative behavior of each sequence:

- For $p = 1.5$, or in general, for $p > 1$, the sequence blows up, i.e., becomes unbounded.

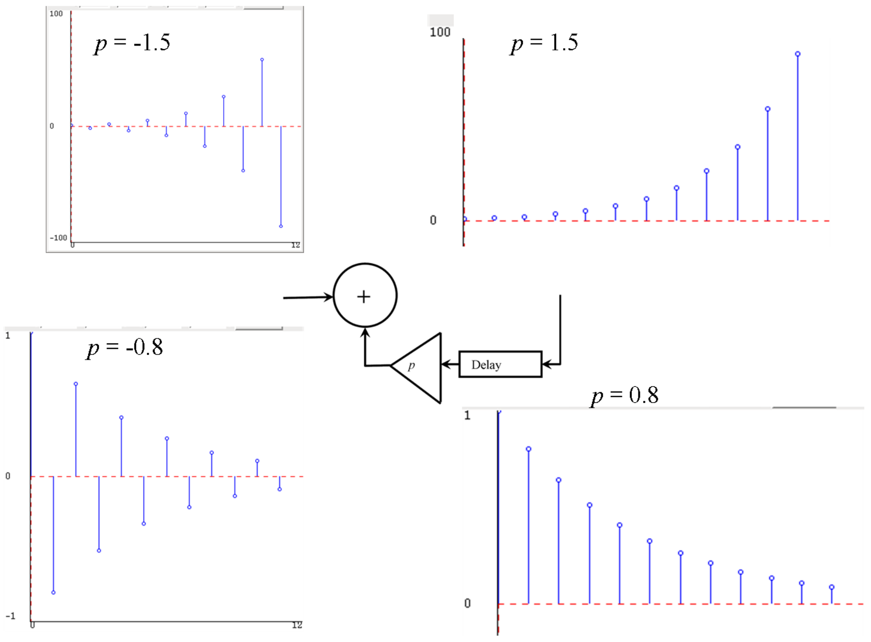


Figure 16: Four different qualitative behaviors—depending on p—of the exponential sequence generated by the simple LTI system shown.

- For $p = 0.5$, or in general for $0 < p < 1$ the sequence decays exponentially to 0.
- For $p = -1.5$, or in general for $p < -1$ the sequence also blows up, oscillating between positive and negative with larger and larger oscillations.
- For $p = -0.5$, or in general for $-1 < p < 0$ the sequence oscillates, decaying to 0.

If we call the behaviors that blow up *unstable* and the ones that decay to constant values *stable*, then we have four different types of qualitative behavior depending, on whether the sequence blows up and whether or not it oscillates. Which type we get depends on p. Stability depends on whether the magnitude of p is less than 1: $|p| < 1$. Oscillation depends on whether p is positive or negative.

Let's interpret this same thing from the system function perspective. Our system function is $1/(1 - p\mathcal{R})$. If we apply the series expansion

$$\frac{1}{1-s} = 1 + s + s^2 + s^3 + \cdots$$

we can write the system function as

$$H = \frac{1}{1 - p\mathcal{R}} = 1 + p\mathcal{R} + p^2\mathcal{R}^2 + p^3\mathcal{R}^3 + \cdots$$

(Don't get nervous about the infinite series, it's just algebra.[7]) In terms of signals, our assumption that the input is 1 for $n = 0$ and and 0 for all other n, says that the input signal x is the unit sample δ. So the output signal is

$$y = Hx = H\delta = 1 + p\mathcal{R}\delta + p^2\mathcal{R}^2\delta + p^3\mathcal{R}^3\delta + \cdots$$

[7]Add a footnote about convergence of infinite series.

Remembering that $R^n\delta$ is the nth-fold delayed unit sample, i.e., the signal that is 1 at n and 0 everywhere else, we see that our output y is just what we computed above: a signal whose value at n is p^n, for all $n \geq 0$.

In terms of the system function $H = 1/(1 - p\mathcal{R})$ the factor p that determines the qualitative behavior is the factor that multiplies $\mathcal{R}$ in the denominator, that is, the factor by which the $\mathcal{R}$ term grows when we do the series expansion. This value is called the *pole* of the system function. So restating what we saw above, the qualitative behavior of the system—stable, unstable, oscillating, non-oscillating—is determined by the pole of the system function. In particular, the result is stable if the magnitude of p is less than 1 and oscillation depends on the sign of p.

We'll see in the next chapter is that this result generalizes to all LTI systems: *Any* system function

$$H = \frac{y}{x} = \frac{b_0 + b_1\mathcal{R} + \cdots + b_j\mathcal{R}^j}{a_0 + a_1\mathcal{R} + \cdots + a_k\mathcal{R}^k}$$

can be rewritten so that the denominator is a product of the form

$$A(1 - p_1\mathcal{R})(1 - p_2\mathcal{R})\cdots(1 - p_k\mathcal{R})$$

The p_i are the *poles* of the system, and stability of the system *for any input* depends on the magnitude of the poles. The system will be stable if all poles have magnitude less than 1, and otherwise not. The one catch here is that the poles will in general be *complex numbers.*

So to sum up our preview: We'll see in the next chapter that for any LTI system, its qualitative behavior can be determined by finding the poles of the system function. Stability depends on whether all poles have magnitudes less than 1, and the locations of the poles in the complex plane determine the size and frequency of oscillations.