MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Fall Semester, 2007

Lecture Notes

Difference Equations and Z Transforms

Zee secret is in zee trunsforms.

Difference Equations with Input

So far, we've used difference equations to model the behavior of systems whose values at some time depend only on their own values at some previous time points. But it is also important to consider systems that depend on an input value, as well.

Let's get the idea by considering a very simple example to start.

Example 1 *Let's think about a simple first-order system with a constant input. We can think, for instance, of a bank account, like Zelda's or Oswald's, into which a constant payment* c *is deposited each year.*

We would model that system using the difference equation

$$y(n) = \alpha y(n-1) + c \quad .$$

Because of the input, c, it is called a *non-homogenous* difference equation. Because it is so simple, we can see what's going on just by expanding it out:

$$\begin{aligned} y(n) &= \alpha y(n-1) + c \\ &= \alpha(\alpha y(n-2) + c) + c \\ &= \alpha(\alpha(\alpha y(n-3) + c) + c) + c \\ &\cdots \\ &= \alpha^n y(0) + c\sum_{i=0}^{n} \alpha^i \\ &= \alpha^n y(0) + c\frac{1-\alpha^{n+1}}{1-\alpha} \quad . \end{aligned}$$

That last step is the standard formula for the sum of a geometric series.

What will happen to this bank account as n goes to infinity? It's clear that if $|\alpha| > 1$ then the first term will go to positive or negative infinity, and we needn't bother thinking about the second term. However, if $|\alpha| < 1$, then as n goes to infinity, the whole expression goes to $c/(1-\alpha)$.

So, for example, if Uncle Oswald lived forever, with a \$100/year being deposited into his account, which as you may recall, had a 5% management fee, the *steady state* value of the account would be $100/(1-0.95) = 2000$.

More generally, a linear difference equation with input can be described in the form

$$\sum_{k=0}^{K} a_k y(n+k) = \sum_{l=0}^{L} b_l x(n+l) \ . \tag{1}$$

We can think about it as a process by which a sequence $x(n)$ is transformed into a new sequence $y(n)$. If $x(n) = 0$ for all n, then this is one of our old familiar *homogeneous* (without input) difference equations from last time, but written slightly differently. To convert back into that form, we'd have to say

$$y(n) = -\sum_{k=1}^{K} \frac{a_{K-1}}{a_k} y(n-k) \ .$$

For the study of the behavior of more complex systems, we'll find it algebraically easier to write difference equations in the form of equation 1.

For difference equations with inputs, natural frequencies play an important role, and can be used to compute solutions. The general solution to a linear difference equation has two parts, one of which depends only on the initial conditions, and one of which depends on the input. The details of how to derive a complete closed-form solution are cool, but more detail than we want to get into in this course. We are going to continue to concentrate on the qualitative behavior of systems described by difference equations, in particular understanding whether or not a given system will be stable in the sense made precise by the definition below.

Definition 1 *A system is* bounded-input bounded-output (BIBO) stable *if* $x(n)$ *being bounded for all* n *necessarily implies that* $y(n)$ *will also be bounded for all* n.

For linear difference equations, If the *natural frequencies* (roots of the characteristic polynomial) λ_i associated with the difference equation are all such that $|\lambda_i| < 1$, then the associated system is BIBO-stable. So, our first step in understanding the behavior of a system, with or without input, is to determine the magnitude of the system's natural frequencies. In the format of equation 1, the natural frequencies are the roots of the characteristic polynomial

$$\sum_{k=0}^{K} a_k \lambda^k = 0 \ .$$

Abstraction and modularity

We've introduced two kinds of objects in our informal discussion above: *sequences* and *transformations on sequences*. As we build complex control or signal-processing systems, or wish to analyze Aunt Zelda's secret financial empire of linked companies, investments, and bank accounts, we need to develop a system of modularity and abstraction so we can put small pieces together into a clearly understood and analyzable system.

We will restrict our attention to a limited but powerful class of sequences, thoses which are solutions to difference equations with input sequences which are bounded. We can start by defining a set of primitive operations on sequences, ones which guarantee that there is a difference equation that relates the given input, usually denoted as $x(n)$, to the final output, usually denoted $y(n)$. These primative operations are:

- *Addition*: $y(n) = x_1(n) + x_2(n)$
- *Scaling*: $y(n) = kx(n)$
- *Shifting back*: $y(n) = x(n-1)$
- *Shifting forward*: $y(n) = x(n+1)$

Note that the general difference equation in (1) can be generated by a combination of scaling, shifting, and adding. Regardless of whether we are referring to one of the primative operations, or to a general difference equation, we think of a *system* as taking an input sequence and producing an output sequence.

When representing more complicated systems, two primary *methods of combination* for systems are quite helpful:

- *Cascade:* Using the output sequence of one system as the the input to another system,
- *Parallel sum:* Summing the sequences generated by two different systems, to generate an output

In the next sections, we'll be able to define this all much more formally.

The important thing here is that when we combine systems, we get another system, and that system has the property that the relationship between the input sequence and the output sequence is describable by a linear difference equation.

Z Transforms

In the Coyote and Roadrunner example from the last lab, we had to play with two coupled linear difference equations. We made it all work out, but it was a lot of algebra. We could think of that as a cascade of two systems, with the output of one (the coyote population) being treated as input to the other (the roadrunner population) and vice versa. As we want to build ever more complex systems, the algebra will get even more complicated, and not be any fun.

Remember how we made multiplication of complex numbers a lot easier by changing to the complex exponential representation? It turns out that we can make operations on sequences a lot easier represent by changing the sequence representation, using something called the *z transform* (also known as generating functions in much of the computer science literature). The z-transform representation of a sequence is no weaker or stronger than the sequence representation: a sequence has has exactly one representation as a z transform, and every power series representation of a z transform corresponds to exactly one sequence. It's easier to calculate values of the generated by a system using the difference equation representation, and we will see that it is easier to combine sequences and operate on them using the z-transform representation.

So here we go. The official z-transform definition:

Definition 2 *Let* $x(n)$ *be a sequence. The* bilateral Z-transform *of* $x(n)$ *is the function*

$$\ddot{X}(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n} \ .$$

What is this about? If we picked a particular z, then this would just be a number, which summed up the power series for that z. But the number wouldn't be a unique representation of that series, because there are other series that could have the same sum for that particular value of z. But if