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

Lecture 6 Notes

Linear Systems 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

$$y[n] = \alpha y[n-1] + c \ .$$

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:

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

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

$$\sum_{k=0}^{M} a_k y[n+k] = \sum_{l=0}^{N} b_l x[n+l] \quad . \tag{1}$$

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

$$y[n] = -\sum_{i=1}^{M} \frac{a_{m-i}}{a_m} y[n-1] \quad .$$

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

Even in more complex problems, it will still be the case that the solution to the difference equation has two parts, one of which depends 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 the system, in particular understanding whether or not it will be stable.

Definition 1 *A system is* bounded-input bounded-output (BIBO) stable *if, whenever* $x[n]$ *is bounded, for all* n*, then* $y[n]$ *is also bounded for all* n.

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 understand its natural frequencies. In the format of equation 1, the characteristic polynomial is

$$\sum_{i=0}^{M} a_i \lambda^i = 0 \quad .$$

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, which are defined by linear difference equations with input. We can define a set of primitive operations on those sequences, which guarantee that if the input sequences are describable in terms of linear difference equations, then so are the results:

- *Addition*: $y[n] = x_1[n] + x_2[n]$
- *Scaling*: $y[n] = kx[n]$
- *Shifting back*: $y[n] = x[n-1]$

- *Shifting forward*: $y[n] = x[n + 1]$

We will call operations on sequences *system functions*.

If these were the only operations we could do, we wouldn't be able to get very far. In fact, we can take these primitive operations, and combine them using two primary *methods of combination*:

- *Cascade:* Feeding the output of one system function into the input of another
- *Parallel sum:* Feeding the input into two different system functions, adding the results, and letting that be the output

(See pictures in lecture slides.) In the next sections, we'll be able to define this all much more formally.

The important thing here is that when we combine system functions, we get another system function, and that system function 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 signals and system functions a lot easier by changing their representation, using something called the *z transform*. The z-transform representation of a linear system is no weaker or stronger than the difference equation representation: each linear difference equation has exactly one representation as a z transform, and vice versa. It's easier to calculate values of the system using the difference equation representation, and easier to combine sequences and operate on them using the z-transform representation.

So. Here we go. The official definition:

Definition 2 *Let* $x[n]$ *be the coefficients of a power series in a variable called* z. *The* bilateral Z-transform *of* $x[n]$ *is the function*

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

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 it wouldn't be a unique representation of that series, because there are other series that could have the same sum for that particular z. But if we were to specify the value the sum for every single possible z, that would exactly nail down our entire infinite series. So, we've traded an infinitely long series for a function defined on an infinite domain. It doesn't necessarily seem like much of a deal. But we'll see it was a good move.