Lecture V: Linear difference and differential equations

Maxim Raginsky

BME 171: Signals and Systems
Duke University

September 10, 2008

This lecture

Plan for the lecture:

1. Discrete-time systems
   - linear difference equations
   - autoregressive moving-average (ARMA) filters
   - recursive implementation of ARMA filters
   - analytical solution of linear difference equations
   - ZIR/ZSR decomposition
2. Continuous-time systems
   - linear differential equations
   - ZIR/ZSR decomposition
   - numerical solution by discretization

Linear difference equations

A wide variety of discrete-time systems are described by **linear difference equations**:

$$y[n] + \sum_{i=1}^{N} a_i y[n-i] = \sum_{i=0}^{M} b_i x[n-i], \qquad n = 0, 1, 2, \ldots$$

where the coefficients $a_1, \ldots, a_N$ and $b_0, \ldots, b_M$ do not depend on n. In order to be able to compute the system output, we also need to specify the **initial conditions (IC's)**

$$y[-1], y[-2], \ldots, y[-N]$$

Systems of this kind are

- linear time-invariant (LTI): easy to verify by inspection
- causal: the output at time n depends only on past outputs $y[n-1], \ldots, y[n-N]$ and on current and past inputs $x[n], x[n-1], \ldots, x[n-M]$

Autoregressive moving-average (ARMA) filters

Systems of this kind are also called **AutoRegressive Moving-Average (ARMA) filters**. The name comes from considering two special cases.

1. **autoregressive filter** of order N, AR(N): $b_0 = \ldots = b_M = 0$

$$y[n] + \sum_{i=1}^{N} a_i y[n-i] = 0, \qquad n = 0, 1, 2, \ldots$$

   In the AR case, the system output at time n is a linear combination of N past outputs; need to specify the ICs $y[-1], \ldots, y[-N]$.

2. **moving-average filter** of order M: $a_1 = \ldots = a_N = 0$

$$y[n] = \sum_{i=0}^{M} b_i x[n-i], \qquad n = 0, 1, 2, \ldots$$

   In the MA case, the system output at time n is a linear combination of the current input and M past inputs; no need to specify ICs.

An **ARMA(N, M)** filter is a combination of both.

Recursive solution of ARMA filter equations

In general, the output of an ARMA filter due to a given input can be computed recursively using the ICs.

$$y[n] + \sum_{i=1}^{N} a_i y[n-i] = \sum_{i=0}^{M} b_i x[n-i], \qquad n = 0, 1, 2, \ldots$$

$$\text{ICs:} \quad y[-1], \ldots, y[-N]$$

Let us first rearrange the system equation:

$$y[n] = -\sum_{i=1}^{N} a_i y[n-i] + \sum_{i=0}^{M} b_i x[n-i].$$

Start at $n = 0$:

$$y[0] = \underbrace{-\sum_{i=1}^{N} a_i y[-i]}_{\text{dep. on the ICs}} + \underbrace{\sum_{i=0}^{M} b_i x[-i]}_{\text{dep. on inputs } x[0],\ldots,x[-M]}$$

Recursive solution of ARMA filter equations: cont'd

Now let $n = 1$:

$$\begin{aligned} y[1] &= -\sum_{i=1}^{N} a_i y[1-i] + \sum_{i=0}^{M} b_i x[1-i] \\ &= -a_1 y[0] - \underbrace{\sum_{i=1}^{N-1} a_{i+1} y[-i]}_{\text{dep. on the ICs}} + \underbrace{\sum_{i=0}^{M} b_i x[1-i]}_{\text{dep. on inputs } x[1],\ldots,x[-M+1]} \end{aligned}$$

Let $n = 2$:

$$\begin{aligned} y[2] &= -\sum_{i=1}^{N} a_i y[2-i] + \sum_{i=0}^{M} b_i x[2-i] \\ &= -a_1 y[1] - a_2 y[0] - \underbrace{\sum_{i=1}^{N-2} a_{i+2} y[-i]}_{\text{dep. on the ICs}} + \underbrace{\sum_{i=0}^{M} b_i x[2-i]}_{\text{dep. on inputs } x[2],\ldots,x[-M+2]} \end{aligned}$$

etc.

Recursive solution of ARMA filter equations: cont'd

In general, to compute the output of an ARMA(N, M) filter at time n, we need the outputs at times $n-1, n-2, \ldots, n-N$ and the inputs at times $n, n-1, \ldots, n-M$.

Implementation complexity:

- memory — at any time, need to store N output values and $M+1$ input values, for a total of $N+M+1$ values
- operations — at any time n, the number of operations need to compute $y[n]$ can be determined like this:

$$y[n] = \underbrace{-\sum_{i=1}^{N} a_i y[n-i]}_{N-1 \text{ add., } N \text{ mult.}} \underbrace{+}_{1 \text{ add.}} \underbrace{\sum_{i=0}^{M} b_i x[n-i]}_{M \text{ add., } M+1 \text{ mult.}} .$$

  So wee need $N+M$ additions and $N+M+1$ multiplications, for a total of $2(N+M)+1$ operations.

These considerations are important when implementing systems like this in software or in hardware. Computational complexity is proportional to $L = N+M$ and is *independent* of n.

Analytical solution of linear difference equations

When the difference equation is simple, we can actually solve the corresponding recursion.

Example: $y[n] + ay[n-1] = x[n], \qquad n = 0, 1, 2, \ldots$

with the IC specifying $y[-1]$. Then

$$\begin{aligned}
y[0] &= -ay[-1] + x[0]; \\
y[1] &= -ay[0] + x[1] \\
&= -a(-ay[-1] + x[0]) + x[1] \\
&= a^2 y[-1] - ax[0] + x[1]; \\
y[2] &= -ay[1] + x[2] \\
&= -a(a^2 y[-1] - ax[0] + x[1]) + x[2] \\
&= -a^3 y[-1] + a^2 x[0] - ax[1] + x[2]; \\
y[3] &= -ay[2] + x[3] \\
&= -a(-a^3 y[-1] + a^2 x[0] - ax[1] + x[2]) + x[3] \\
&= a^4 y[-1] - a^3 x[0] + a^2 x[1] - ax[2] + x[3]
\end{aligned}$$

Analytical solution of linear difference equations: cont'd

The pattern emerges:

$$y[n] = (-a)^{n+1} y[-1] + \sum_{i=0}^{n} (-a)^{n-i} x[i], \qquad n = 0, 1, 2, \ldots$$

Note that $y[n]$ a sum of two terms:

1. $(-a)^{n+1} y[-1]$ depends only on the IC, but not on the input
2. $\sum_{i=0}^{n} (-a)^{n-i} x[i]$ depends only on the input, but not on the IC

This decomposition holds generally for any ARMA system: the system output at time n is a sum of the AR-only and the MA-only outputs at time n.

Decomposition of ARMA filter outputs

Consider an ARMA(N, M) system

$$y[n] = -\sum_{i=1}^{N} a_i y[n-i] + \sum_{i=0}^{M} b_i x[n-i], \qquad n = 0, 1, 2, \ldots$$

with the ICs $y[-1], \ldots, y[-N]$. Then we can write the output at time n as

$$y[n] = y_0[n] + y_1[n]$$

where $y_0[n]$ and $y_1[n]$ satisfy the equations

$$y_0[n] = -\sum_{i=1}^{N} a_i y_0[n], \qquad n = 0, 1, 2, \ldots$$

with the ICs $y_0[-1] = y[-1], \ldots, y_0[-N] = y[-N]$, and

$$y_1[n] = -\sum_{i=1}^{N} a_i y_1[n-i] + \sum_{i=0}^{M} b_i x[n-i]$$

with the ICs $y_1[-1] = \ldots = y_1[-N] = 0$.

Decomposition of ARMA filter outputs: cont'd

Note that $y_0[n]$ is the output of the system determined by the ICs only (setting the input to zero), while $y_1[n]$ is the output of the system determined by the input only (setting the ICs to zero).

To see that $y[n] = y_0[n] + y_1[n]$, let $z[n] = y_0[n] + y_1[n]$. Then

$$\begin{aligned} z[n] &= y_0[n] + y_1[n] \\ &= -\sum_{i=1}^{N} a_i y_0[n-i] - \sum_{i=1}^{N} a_i y_1[n-i] + \sum_{i=0}^{M} b_i x[n-i] \\ &= -\sum_{i=1}^{N} a_i \Big(\underbrace{y_0[n-i] + y_1[n-i]}_{z[n-i]} \Big) + \sum_{i=0}^{M} b_i x[n-i] \end{aligned}$$

with the ICs $z[-1] = y[-1], \ldots, z[-n] = y[-N]$. Thus, $z[n]$ satisfies the same system equation as $y[n]$ with the same ICs.

Decomposition of ARMA filter outputs: cont'd

Consider the output decomposition $y[n] = y_0[n] + y_1[n]$ of an ARMA(N, M) filter

$$y[n] = -\sum_{i=1}^{N} a_i y[n-i] + \sum_{i=0}^{M} b_i x[n-i], \qquad n = 0, 1, 2, \ldots$$

with the ICs $y[-1], \ldots, y[-N]$. $y_0[n]$ is often called the **zero-input response** (ZIR) of the filter (referring to the fact that it is determined by the ICs only), while $y_1[n]$ is called the **zero-state response** (ZSR) of the filter (referring to the fact that it is determined by the input only, with the ICs set to zero). Thus, the output of an ARMA filter at time n is the sum of the ZIR and the ZSR at time n.

Linear differential equations

Now we switch to continuos-time systems. A wide variety of continuous-time systems are described the **linear differential equations**:

$$y^{(N)}(t) + \sum_{i=0}^{N-1} a_i y^{(i)}(t) = \sum_{i=0}^{M} b_i x^{(M-i)}(t), \qquad t \geq 0$$

where $y^{(i)}(t)$ denotes the ith derivative of $y(t)$, and $y^{(0)}(t) = y(t)$. Just as before, in order to solve the equation for $y(t)$, we need the ICs. In this case, the ICs are given by specifying the value of y and its derivatives 1 through $N-1$ at $t = 0^-$ (time "just before" $t = 0$):

$$y(0^-), y^{(1)}(0^-), \ldots, y^{(N-1)}(0^-).$$

Note: the ICs are given at $t = 0^-$ to allow for impulses and other discontinuities at $t = 0$.

Systems described in this way are

1. linear time-invariant (LTI): easy to verify by inspection
2. causal: the value of the output at time t depends only on the output and the input at times $0 \leq \tau \leq t$

Decomposition of the solution

Just as in the discrete-time case, the solution $y(t)$ of

$$y^{(N)}(t) + \sum_{i=0}^{N-1} a_i y^{(i)}(t) = \sum_{i=0}^{M} b_i x^{(M-i)}(t), \qquad t \geq 0$$

with the ICs $y(0^-), y^{(1)}(0^-), \ldots, y^{(N-1)}(0^-)$ can be decomposed as

$$y(t) = y_0(t) + y_1(t)$$

where the **zero-input response** (ZIR) $y_0(t)$ satisfies the equation

$$y_0^{(N)}(t) + \sum_{i=0}^{N-1} a_i y_0^{(i)}(t) = 0, \qquad t \geq 0$$

with the ICs $y(0^-), y^{(1)}(0^-), \ldots, y^{(N-1)}(0^-)$, while the **zero-state response** (ZSR) $y_1(t)$ satisfies the equation

$$y_1^{(N)}(t) + \sum_{i=0}^{N-1} a_i y_1^{(i)}(t) = \sum_{i=0}^{M} b_i x^{(M-i)}(t), \qquad t \geq 0$$

with the ICs $y_1(0^-) = y_1^{(1)}(0^-) = \ldots = y_1^{(N-1)}(0^-) = 0$.

ZIR and ZSR

Mathematicians use the names **homogeneous solution** and **particular solution** instead of ZIR and ZSR.

A standard method for obtaining the ZIR is to try a solution of the form

$$y_0(t) = \sum_{i=1}^{N} C_i e^{\lambda_i t}$$

and then use the ICs to solve for $C_1, \ldots, C_N$ and $\lambda_1, \ldots, \lambda_N$.

To obtain the ZSR, one would have to use the method of integrating factors, described in every textbook on differential equations. Later in this course, we will look at alternative ways of solving for the ZSR using the Laplace transform.

Discretization

Linear differential equations can be solved numerically on a computer by converting them into linear difference equations.

Example: $y'(t) + ay(t) = bx(t)$

Let us pick some $T > 0$. Then for each $n = 0, 1, 2, \ldots$ we have

$$y'(nT) + ay(nT) = bx(nT)$$

Assuming T is sufficiently small, we can approximate

$$y'(nT) \approx \frac{y(nT + T) - y(nT)}{T}$$

This is called **Euler's approximation** of the first derivative. Now define discrete-time signals $x[n]$ and $y[n]$ via

$$x[n] = x(nT) \qquad \text{and} \qquad y[n] = y(nT).$$

Discretization: cont'd

Then we approximate our original problem by

$$\frac{y[n+1]-y[n]}{T}+ay[n]=bx[n], \qquad n=0,1,2,\ldots$$

Making the replacement $n \longrightarrow n-1$, we obtain the discrete-time system

$$y[n]+(aT-1)y[n-1]=bTx[n-1], \qquad n=0,1,2,\ldots$$

which we can now implement recursively, given the (discretized) ICs.

The same principle is used to discretize higher-order differential equations. For example, we can approximate

$$\begin{aligned} y''(nT) &\approx \frac{y'(nT+T)-y'(nT)}{T} \\ &\approx \frac{1}{T}\left(\frac{y((n+1)T+T)-y((n+1)T)}{T}-\frac{y((n+1)T)-y(nT)}{T}\right) \\ &= \frac{y((n+2)T)-2y((n+1)T)+y(nT)}{T^2}. \end{aligned}$$