Signals

In 6.01, a signal is a sequence of discrete values, indexed by time. Signals are labeled by uppercase letters and values of a signal at a specific point are labeled by lowercase letters with an index.

Unit sample:

\[ b[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \]

Unit step:

\[ u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} \]

Systems

Systems transform signals, changing one signal into another. There are many ways to describe systems:

- Block diagrams
- Difference equations
- Operator equations
- System functionals
- State machines

System building blocks:

**Gain**

\[ X \xrightarrow{k} Y = kX \]

\[ y[n] = k \cdot x[n] \]

**Delay**

\[ X \xrightarrow{R} Y = RX \]

\[ y[n] = x[n-1] \]
Block diagrams:

- Shows "flow" of system components.
- Makes it easy to label parts of the system.

Difference equation:
\[ y[n] = x[n] + Ky[n-1] \]
- Gives a "formula" for computing a discrete value from other discrete values.

Operator equation:
\[ Y = X + KR\dot{Y} \]
- Manipulates entire signals, rather than single values, as in a difference equation.
- Can use normal algebra.

System functionals:
\[ \frac{Y}{X} = \frac{1}{1-KR} = H \]
- Solve operator equations for \( \frac{Y}{X} \)
- Often labeled "H" for \( \frac{Y}{X} \)
- Makes it easy to find poles.
- Lets systems be "black-boxed".

State machines:
- State machines formalize the concept of system memory, or state.
  - System behavior depends on
    - Current input
    - Current state
- From this, the system generates a new state, and some output.
- In our state machine class, there are two methods that define a system:
  - getStartState(self) - returns an initial state.
  - getNextValues(self, state, input) - Based on current input and state (input arguments), return output and next state.
The SM class has other functions that do not need to be overridden:
- `start(self)` - initializes a state machine
- `step(self, inp)` - applies one input to the state machine
- `transduce(self, inps)` - applies a sequence of inputs to the state machine.

How can we implement our system as a state machine?
- We need to decide on a state representation.
  - From difference equation, all we need is
    - `x[n]` - current input
    - `y[n-1]` - last output
  - So, state is just `y[n-1]`

```python
class ExampleSystem(SM):
    def get_start_state(self):
        return 0  # start from rest
    def get_next_value(self, state, input):
        output = input + 2 * state
        newstate = output
        return (output, newstate)
```

State machines can represent any LTI system. State machines can also represent other processes as well, such as the ribosone SM or the Farmer-Cabbage-Wolf SM you explored in homeworks.
LTI

Linear:
\[ X \rightarrow \mathbf{H} \rightarrow Y \]
\[ X' \rightarrow \mathbf{H} \rightarrow Y' \]
\[ ax + bx' \rightarrow \mathbf{H} \rightarrow ay + by' \]

Time invariant:
- Delay the input, delay the output
\[ RX \rightarrow \mathbf{H} \rightarrow RY \]

Response of a system

- The output of a system when the input is a unit sample signal is known as the response.

- What is the response to our system?
\[
\begin{align*}
y[0] &= x[0] + Ky[-1] = 1 + K \cdot 0 = 1 \\
&\vdots \\
y[n] &= K^n 
\end{align*}
\]

- \( K \) is what is known as a "pole."

- The response of a system with a single pole is proportional to that pole raised to the \( n \)th power, as shown in the example.
Finding poles:
- In the system functional, replace \( R \) with \( 1/z \).
- Find the roots of the denominator (values of \( z \) where the system functional \( \to \pm \infty \)).

From the example system:

\[
H = \frac{1}{1 - KR} \quad \Rightarrow \quad \frac{1}{1 - \frac{1}{2}K} = \frac{2}{z - K} \quad \Rightarrow \text{pole is at } z = K
\]

- If the system is more complicated, the denominator can have multiple roots.
  * We say the system has multiple poles.
  * In general, these can be complex.

- A system with multiple poles can always be shown to be the sum of multiple single pole systems via partial fraction decomposition.
  * See book, section 5.5.2.
Complex numbers
- Complex numbers are typically represented by a real part and an imaginary part: \( a + bj \)
- We can plot these in 2D:

![Diagram of a complex number in the complex plane with real and imaginary axes]

- This suggests another way we might represent these values, as a magnitude and an angle.

- The magnitude is just distance: \( A = \sqrt{a^2 + b^2} \)
- The angle can be found by the arctangent: \( \arctan\left( \frac{b}{a} \right) \)
  - In Python, we use \( \text{atan2}(b, a) \) in order to handle quadrants correctly.

Polar form:
- Remember the unit circle:
  \[
  \cos \theta = \text{x position} = \text{real part} \\
  \sin \theta = \text{y position} = \text{imaginary part}
  \]

- \( a + bj = A(e^{i\theta}) \)
- Euler's formula: \( \cos \theta + jsin \theta = e^{i\theta} \)

What happens when we raise complex numbers to powers?

\[
(a + bj)^n = \text{ugly} \\
(Ae^{i\theta})^n = A^n e^{i(n\theta)}
\]

\[\text{Angle increases linearly} \]
\[\text{Amplitude increases / decreases geometrically}\]
Some examples:

\[ 0.9 + 0j \rightarrow A = 0.9, \quad \theta = 0 \]

\[
\begin{align*}
0.9^0 & = 1 \\
0.9^1 & = 0.9 \\
0.9^2 & = 0.81 \\
0.9^3 & = 0.72
\end{align*}
\]

\[
\frac{\pi}{2} \left( 1 + 1j \right) \rightarrow A = 1, \quad \theta = \frac{\pi}{4}
\]

\[
\begin{align*}
\left( e^{j\pi/4} \right)^0 & = e^{j0} = 1 \\
\left( e^{j\pi/4} \right)^1 & = e^{j\pi/4} = e^{j\pi/2} (1 + 1j) \\
\left( e^{j\pi/4} \right)^2 & = e^{j\pi} = e^{j2\pi} = 1
\end{align*}
\]

What would happen with \(0.9e^{j\pi/4}\)?

- Magnitude decreases as angle increases
- Spiral

CONVERGENCE

- What happens as \(n \rightarrow \infty\)?

inside the unit circle (\(A < 1\)), it "converges" (approaches 0)

outside the unit circle (\(A > 1\)), it "diverges" (goes towards ±∞)
System responses (again):

- If there are multiple poles, which one matters most?
  - The one that overpowers the rest
  - This will be the pole with the largest magnitude
  - Called "dominant pole"

- What are the types of system responses?

<table>
<thead>
<tr>
<th>Dominant pole</th>
<th>Response</th>
<th>Dominant pole magnitude</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real, positive</td>
<td>monotonic</td>
<td>&lt; 1</td>
<td>Convergent</td>
</tr>
<tr>
<td>Real, negative</td>
<td>alternating</td>
<td>= 1</td>
<td>Neither</td>
</tr>
<tr>
<td>Complex</td>
<td>oscillating</td>
<td>&gt; 1</td>
<td>Divergent</td>
</tr>
</tbody>
</table>

- For oscillating responses, we're often interested in the period of oscillation.

  \[ \text{Period} = \frac{2\pi}{\theta} = \frac{\text{total angle}}{\text{angle per timestep}} \]

- If the poles are complex, they always come in conjugate pairs.

- This is why the system output is real.

Control systems

\[ X \rightarrow \text{error} \rightarrow \text{controller} \rightarrow \text{command} \rightarrow \text{plant} \rightarrow Y \]

controller - generates a command signal from an error signal. This is where you, the control designer, have flexibility.

plant - the physics of the system being controlled

sensor - the subsystem that detects the real world
Reviewing the design lab systems:

**DLO 4:**
\[
\begin{align*}
D_o & \to + \quad \text{controller} \quad \text{plant} \\
& \quad \text{sensor}
\end{align*}
\]

System functional: \(-\frac{TRK}{1-R-TR^2K}\)

Root locus:

Poles: \(1-R-TR^2K \rightarrow z^2 - 2 - TK\)
\[z = \frac{1}{2} \pm \frac{\sqrt{1-4TK}}{2}\]

**DLO 5:**

System functional: \(\frac{T^2UKR^2}{(1+T^2UK)(R^2 - 2R + 1)}\)

Root locus:

Poles: \(\frac{z}{2} = \frac{\sqrt{4 - 4(1+T^2UK)}}{2} = 1 \pm j\sqrt{NK}\)

**DLO 6:**

This system uses two nested loops.

This makes it a system with three poles.