6.01 Introduction to EECS I

Lecture 2: Signals and Systems

Lecturer: Adam Hartz (hartz@mit.edu)

As you come in...

- Grab one handout (on the table by the entrance)
- If you plan on using a laptop or smartphone during lecture, please sit near the back.

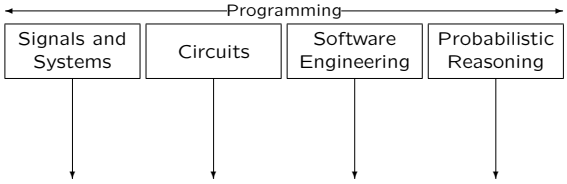
Notes

6.01: Introduction to EECS I

The **intellectual themes** in 6.01 are recurring themes in engineering:

- design of complex systems
- modeling and controlling physical systems
- augmenting physical systems with computation
- building systems that are robust to uncertainty

Approach: focus on **key concepts** to pursue **in depth**



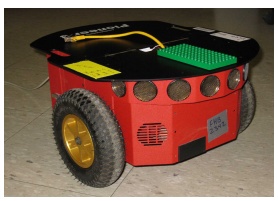
Notes

programming throughout course

6.01: Pedagogy

Most of the learning is in the lab!

- active learning with hands-on exercises
- open-ended problems with multiple correct solutions
- multiple levels of individualized help (partners, LAs, TAs, Instructors)



Intellectual themes are developed in context of a mobile robot.

Notes

6.01 Structure

- **Lecture:** Mon, 9:30a-11:00a, in 26-100
- **Software Lab:** Mon/Tue, 90 minutes, in 34-501
  - practice with material from lecture and readings
  - preparation for design lab
  - useful self-check of understanding
- **Design Lab:** Wed/Thu/Fri, 3 hours, in 34-501
  - work with a partner (new partner each week)
  - more open-ended interaction with material
  - checkoffs in lab
- **Nanoquiz:** 15 minute quiz at the start of DL
- **Homework:** Weekly online readings and exercises
  - Released Monday mornings
  - Looking back, but also looking ahead
  - Due Sunday nights (11pm)
- **Exams:** Two 2-hour exams, plus one 3-hour final exam
- **Student Lab Assistant Option**

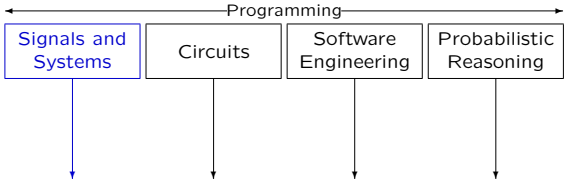
Notes

6.01: Introduction to EECS I

The **intellectual themes** in 6.01 are recurring themes in engineering:

- design of complex systems
- modeling and controlling physical systems
- augmenting physical systems with computation
- building systems that are robust to uncertainty

Approach: focus on **key concepts** to pursue **in depth**

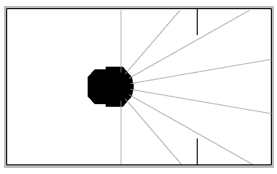


Focus: discrete-time feedback control systems

Notes

Modeling and Analyzing Behavior

Consider the "wall finder" problem from Design Lab 1:



Consider 2 possible "wall finder" behaviors.

Notes

Behavior 1 goes near to the wall and stops

Behavior 2 goes very close to the wall, then oscillates back and forth until stopping

Check Yourself!

Which behavior is better?

1. Behavior 1
2. Behavior 2
3. Both are good
4. Both are bad

Notes

Behavior 1 is more efficient in distance traveled

Behavior 2 is more efficient in time and has higher accuracy, but initially overshoots and undershoots (oscillates)

Modeling and Analyzing System Behavior

Goal: develop a framework for simulating and analyzing behavior

Many pieces:

- mathematical: difference equations, operator notation
- computational: state machines
- analysis: responses, system functionals, poles

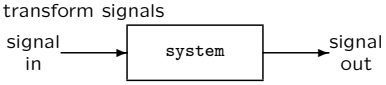
Notes

The Signals and Systems Abstraction

Signals are functions of a single parameter (time, for our purposes)

- signals represented by capital letters: X
- n^{th} sample of X denoted with lowercase: $x[n]$

Systems transform signals



In 6.01, we consider **discrete-time** signals and systems.
In 6.01, we consider **LTI** systems.

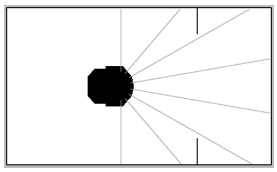
- **linear**: output at time n is a linear combination of previous inputs and outputs (system consists only of delays, gains, and adders)
- **time-invariant**: output of the system does not depend on the absolute time at which the system was started

Notes

LTI: Linear Time Invariant

Analyzing System Behavior

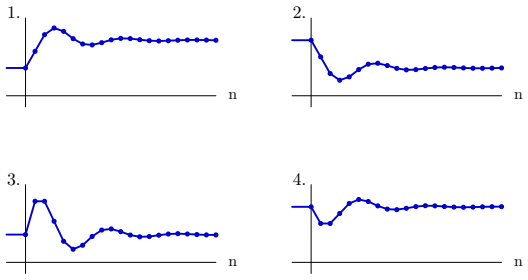
Consider the "wall finder" again.



Notes

Check Yourself!

Which of the following best represents the robot's measured distance for the example just shown?



5. None of the above

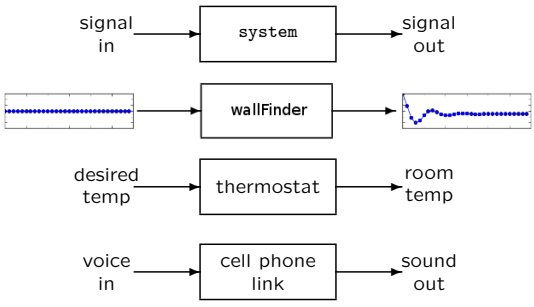
Notes

Behavior 2, graph 2 represents the oscillating movement.

Robot initially moves towards wall, distance decreases, then oscillates to rest.

Analyzing System Behavior

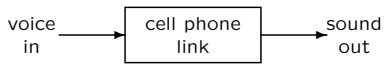
Represent **systems** (physical, math, computational) by the way they transform an **input signal** into an **output signal**.



Notes

Systems are Modular!

Example: cell phone link



Notes

Modularity and Abstraction

Thinking about complicated systems is *complicated*.

Framework for thinking about complicated systems:

- **Primitives**
- Means of **Combination**
- Means of **Abstraction**

Example: Python

- Primitives: `+`, `*`, `==`, `!=`, ...
- Combination: `if`, `while`, `f(g(x))`, ...
- Abstraction: `def`

Notes

Simple Systems

Wire

Block Diagram:

Difference Equation:

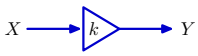
$$y[n] = x[n]$$

Notes

Simple Systems

Gain

Block Diagram:



Difference Equation:

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

Notes

Amplifier

Simple Systems

Delay

Block Diagram:

Difference Equation:

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

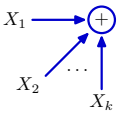
Notes

Simple Systems

Addition

An adder can be thought of as a system which takes arbitrarily-many inputs and outputs their sum.

Block Diagram:



Difference Equation:

$$y[n] = x_1[n] + x_2[n] + \ldots + x_k[n]$$

Notes

Check Yourself!

Consider the system described by:

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

What is the output of this system when its input is the "unit sample" sequence Δ?

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

Notes

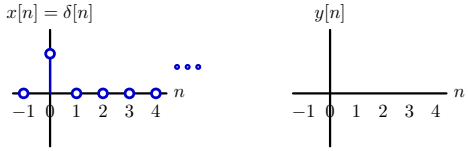
Output (y) = [1, -1, 0, 0, …]

Draw a table of input/out at time steps.

Representations: Difference Equations

Convenient for step-by-step analysis

Find $y[n]$ given $x[n] = \delta[n]$: $y[n] = x[n] - x[n-1]$

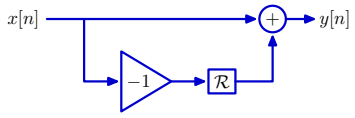


Notes

Representations: Block Diagrams

Graphical representation of the difference equation.

Represent $y[n] = x[n] - x[n-1]$ with a block diagram:

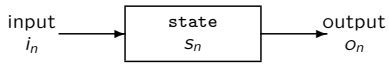


Notes

Computational Framework

A convenient abstraction for modeling computations that evolve with time is the **state machine**.

Like a function, but output depends on **state** (memory) as well as input.



On the n^{th} timestep, the state machine:

- gets input i_n
- generates output o_n
- moves to a new state s_{n+1}

Explicit representation of stepwise nature of time-varying systems.

Notes

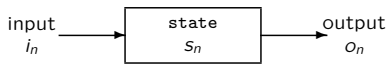
State Machines

Example: accumulator

On each step, the input is added to the accumulated inputs, and the result becomes the new output.

Notes

Accumulator



Notes

Sum of all previous inputs.

State Machines in Python

Using OOP to represent state machines:

SM Class:

- `start()` : initialize the state machine
- `step()` : receive and process new input
- `transduce()` : make repreated calls to `step`

Accumulator subclass of SM:

- `startState`: initial contents of `state` at time 0
- `getNextValues(state, input)`: method to process input

Instances of Accumulator:

- `state`: sum of inputs so far

Notes

SM Class

The generic methods of the SM class use **startState** to initialize the instance variable **state**. Then **getNextValues** is used to process inputs, so that **step** can update **state**.

```
class SM:
    startState = None
    def getStartState(self):
        return self.startState
    def start(self):
        self.state = self.getStartState()
    def step(self, inp):
        (s, o) = self.getNextValues(self.state, inp)
        self.state = s
        return o
    def transduce(self, inps):
        result = []
        self.start()
        for i in inps:
            result.append(self.step(i))
        return result
```


Notes

State Machine (SM) Class

Accumulator

```
class Accumulator(SM):
    startState = 0

    def getNextValues(self, state, inp):
        return (state+inp, state+inp)
```


Notes

Check Yourself!

```
>>> a = Accumulator()
>>> a.start()
>>> a.step(7)
>>> b = Accumulator()
>>> b.start()
>>> b.step(10)
>>> a.step(-2)
>>> print a.state,a.getNextValues(8,13),b.getNextValues(8,13)
```

What will be printed?

1. `5 (18, 18), (23, 23)`
2. `5 (21, 21), (21, 21)`
3. `15 (18, 18), (23, 23)`
4. `15 (21, 21), (21, 21)`
5. None of the above

Notes

So far...

Primitives: Gain, Delay, Adder
Representations: Difference Equation, Block Diagram, `SM`

Notes

From Samples to Signals

"Lumping" all of the (possibly infinite) samples into a single object (the **signal**) simplifies its manipulation.

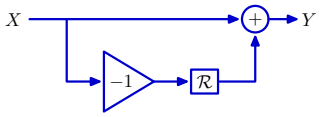
This is analogous to:

- representing coordinates in 3-space as points
- representing lists of numbers as vectors in linear algebra
- creating an object in Python

Notes

From Samples to Signals

manipulate signals rather than individual samples.



Nodes represent whole signals (e.g., X and Y).
Boxes **operate** on those signals:

- Delay: shift whole signal to right by 1 time step
- Add: sum two signals
- -1: multiply by -1

Signals are primitives.
Operators are the means of combination.

Notes

Operator Notation

Systems can now be represented by compact symbolic equations.

Let $\mathcal{R}$ represent the **right-shift operator**:

$$Y = \mathcal{R}\{X\} \equiv \mathcal{R}X$$

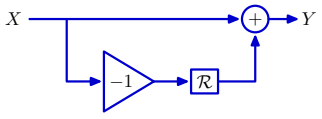
where X represents the whole input signal ($x[n]$ for all n) and Y represents the whole output signal ($y[n]$ for all n)

Notes

Operator Notation

Systems are concisely represented with operators.

Consider:



Equivalent representation with $\mathcal{R}$:

$$Y = X - \mathcal{R}X = (1 - \mathcal{R})X$$

Notes

Check Yourself!

Let $Y = \mathcal{R}X$. Which of the following is/are true?

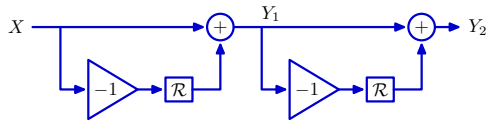
- $y[n] = x[n]$ for all n
- $y[n+1] = x[n]$ for all n
- $y[n] = x[n+1]$ for all n
- $y[n-1] = x[n]$ for all n
- None of the above

Notes

When Y = RX than y[n] = x[n-1]

Operator Algebra

Cascade of systems: apply second system to the output of the first



$$Y_1 = (1 + \mathcal{R})X$$
$$Y_2 = (1 + \mathcal{R})Y_1 = (1 + \mathcal{R})(1 + \mathcal{R})X$$

Notes

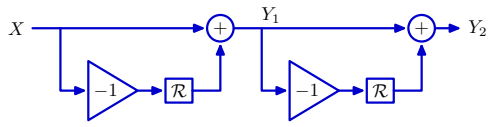
<< Error, should be:

Y1 = (1-R)X = X - RX

Y2 = (1-R)(1-R)X

Operator Algebra

Operator expressions expand and reduce like polynomials!



Difference equation:

$$y_2[n] = y_1[n] + y_1[n-1]$$
$$= (x[n] + x[n-1]) + (x[n-1] + x[n-2])$$
$$= x[n] + 2x[n-1] + x[n-2]$$

Operator notation:

$$Y_2 = (1 + \mathcal{R})(1 + \mathcal{R})X$$
$$= (1 + 2\mathcal{R} + \mathcal{R}^2)X$$

Notes

Operator Algebra

Expressions involving $\mathcal{R}$ obey many familiar laws of algebra:

- Commutativity:

$$\mathcal{R}(1-\mathcal{R})X = (1-\mathcal{R})\mathcal{R}X$$

- Multiplication distributes over addition:

$$\mathcal{R}(1-\mathcal{R}) = \mathcal{R}-\mathcal{R}^2$$

- Associativity:

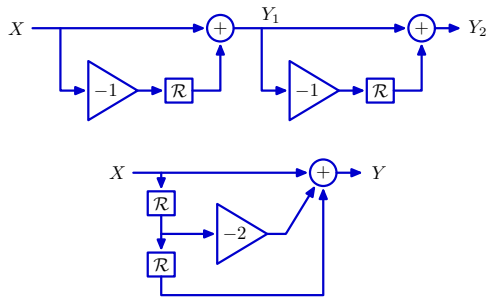
$$(2-\mathcal{R})(\mathcal{R}(1-\mathcal{R})) = ((2-\mathcal{R})\mathcal{R})(1-\mathcal{R})$$

Applies your existing expertise with polynomials to understand block diagrams, and thereby understand systems.

Notes

Operator Approach

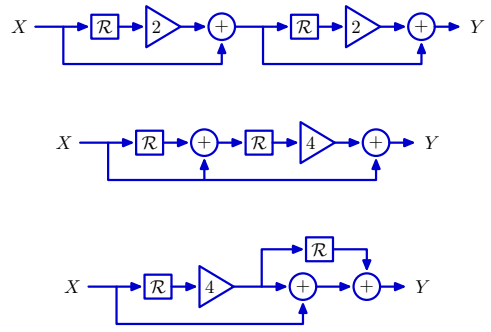
Facilitates seeing relations among systems. The following are "equivalent" in terms of input/output relationship when starting from rest:



Notes

Check Yourself!

How many of the following systems are equivalent?

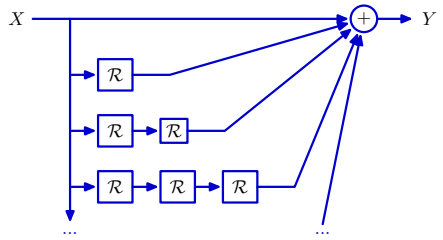


Notes

Accumulator Revisited

Accumulator had a persistent response to transient input! This system could be represented as:

$$Y = (1 + \mathcal{R} + \mathcal{R}^2 + \ldots)X$$

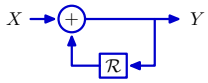


Notes

Accumulator Revisited

Another view of the accumulator was that it added its current input to its previous output. This suggests the relation:

$$Y = X + \mathcal{R}Y$$



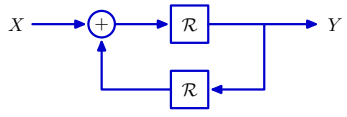
In feedforward LTI systems, the output is a linear combination of previous inputs.

In feedback LTI systems, the output is a linear combination of previous inputs and previous outputs.

Notes

Check Yourself!

Determine the difference equation that relates $x[\cdot]$ and $y[\cdot]$ in the system below:



1. $y[n] = x[n-1] + y[n-1]$
2. $y[n] = x[n-1] + y[n-2]$
3. $y[n] = x[n-1] + y[n-1] + y[n-2]$
4. $y[n] = x[n-1] + y[n-1] - y[n-2]$
5. None of the above

Notes

Linearity

Any system composed of adders, gains, and delays can be represented by a difference equation:

$$y[n] = \sum_{k=0}^{\infty} d_k x[n-k] + \sum_{j=1}^{\infty} c_j y[n-j]$$

Such a system can also be represented by an operator expression:

$$(1 - c_1\mathcal{R} - c_2\mathcal{R}^2 - \ldots)Y = (d_0 + d_1\mathcal{R} + d_2\mathcal{R}^2 + \ldots)X$$

Notes

Impacts of LTI

Superposition:
Any input signal X can be represented as a sum of scaled, shifted unit-sample signals. The output of a system Y can be represented as a sum of unit-sample *responses* scaled and shifted in the same manner.

Example:

$$X = \Delta - 2\mathcal{R}^2\Delta + \mathcal{R}^4\Delta$$

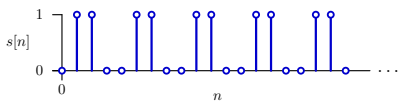
If the system's unit sample response is H, then the output of the system when X is its input will be:

$$Y = H - 2\mathcal{R}^2H + \mathcal{R}^4H$$

Notes

Check Yourself!

Below is the unit *step* response of a system. What is its unit *sample* response?



Notes

Recap

Introduced the signals and systems abstraction.

Introduced several representations of systems:
difference equations, block diagrams, state machines, operator expressions.

Software Lab 2:
Develop `SM` framework for simulating LTI systems

Design Lab 2:
Analysis and control of physical system (wall finder)

Notes

Notes

Notes