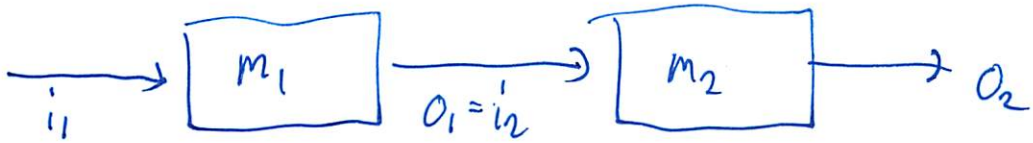
Reading: Combost Abstractions of SMs 4.2

9/25

- instead of build 1 complex state machine
  - combine multiple simple SMs (PCAP)
  - look at data flow
  - sometimes conditional
  - sometimes sequential

Cascade

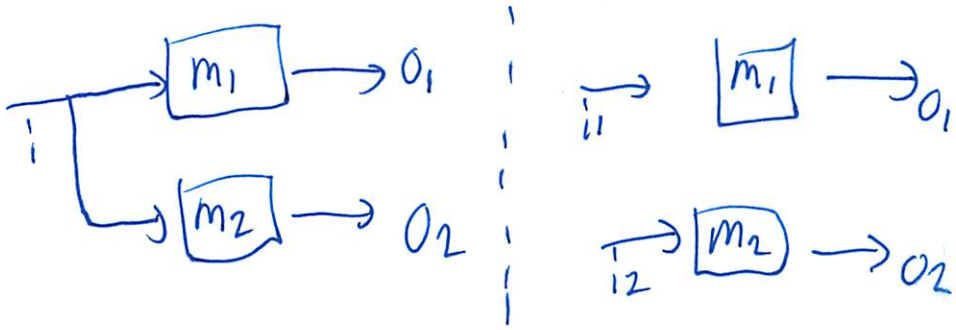


- ie combine 2 delay machines to delay # 2 positions

$$o_2[t] = i_{n_1}[t-2]$$

- order matters of course

Parallel

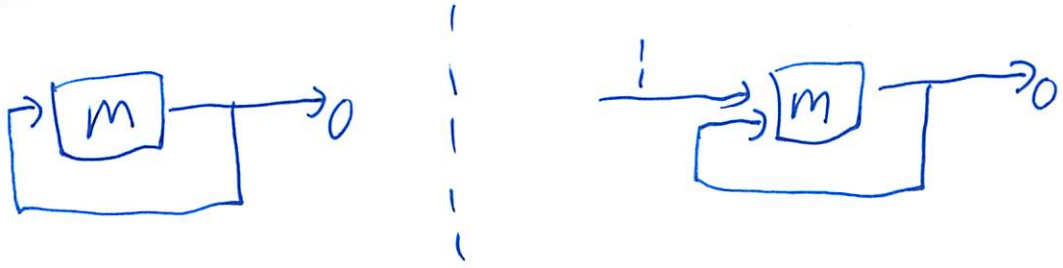


Class Parallel
- initalizes 2 machines
- runs them seperate (in parallel)

if each takes a different input
- then inp is a list/tuple
- and script splits values

(2)

Feedback



- output of machine is fed back in
- in and out must be same format
- example

$S = \#$
$I = \#$
$O = \#$

$n(s,i) = i+1$
$o(s,i) = n(s,i)$
$s_0 = 0$

$O[t] = i[t] + 1$ and $i[t] = o[t]$

* Machine must not have a direct dependence of its output on its input

(so what, must have a delay or what ?? (confused)

- can feed undefined in.
  - because we don't know what input should be (??)
  - so need to add condition to getNextValue method to just pass along "undefined"
  - so going to define SafeAdd + safeMul

(3)

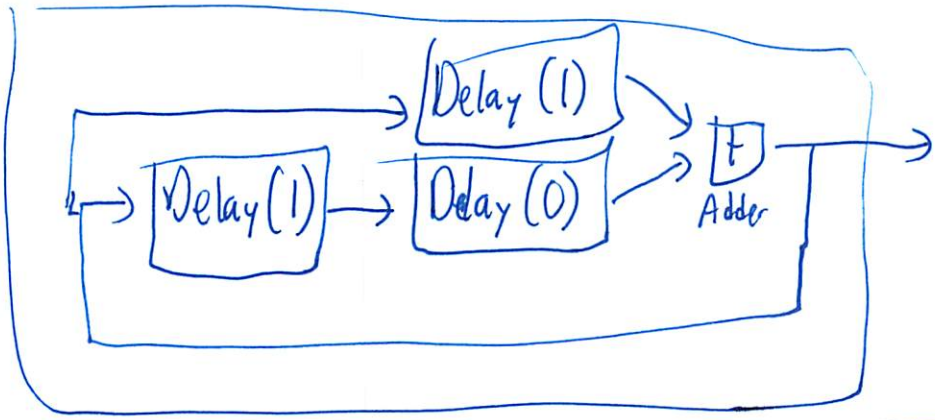
inp is ignored
(duh)

and 0 is fed into the next input

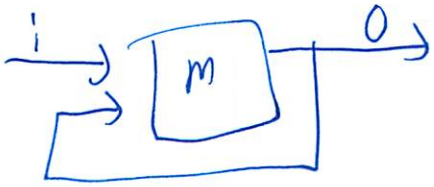
Now can combine w/ an Increment method

Fibonacci

- getting fancy → complex machine out of trivial components
- 1, 1, 2, 3, 5, 8, 13, 21, etc
  ~ Sum of 2 previous inputs



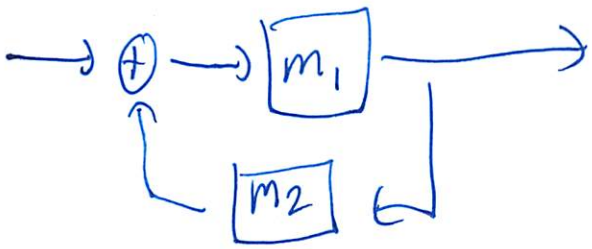
Feedback 2



and they show code to implement
(look at when problem covers)

(4)

Feedback Subtract + Feedback Add



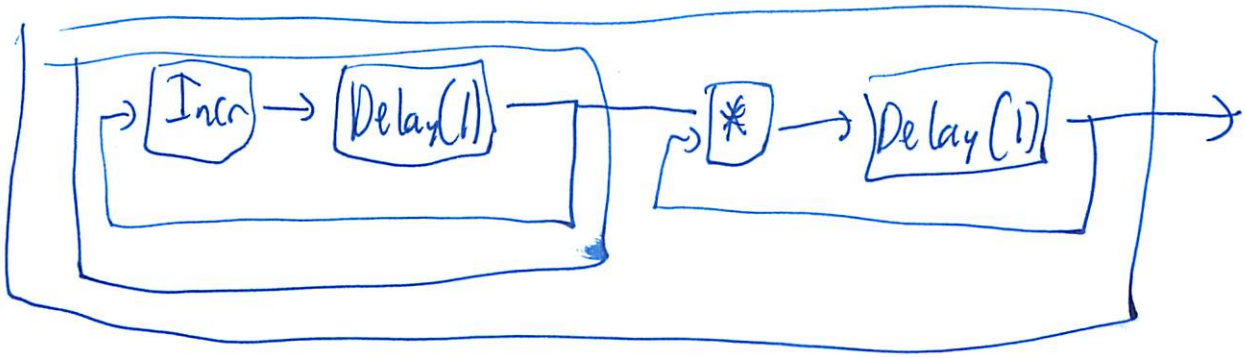
new M = sm. FeedbackAdd (m1, m2)

R(0) — Delay

Wire — just passes through

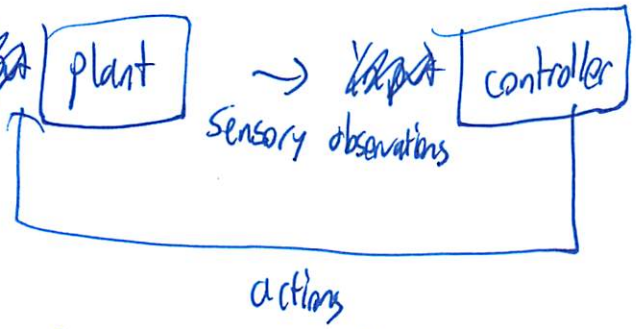
Factorial

- 1!, 2!, 3!.
- multiplying previous values · index



(5)

Plants + Controllers

- couple a "controller" and a so called "plant"
- plant = factory or outside env to control
- 

- like the soar env / robot
- example

$S = \#$
$I = \#$
$O = \#$

$n(s,i) = h(d_{desired} - i)$
$O(s) = s$
$S_0 = d_{init}$

Conditionals

- make choices at runtime

Switch

- runs 2 machines in parallel
- at each input decides which machine to send to

Multiplex

- update both machines each step
- use condition to select which output

If

- only used in complex contexts, skip

(6)

Terminating state machines + seq. compositions

- all machines before will ∞ run
- add I property → to terminate w/ d(s) done function
  - ↳ true if done
  - ↳ false if not

- Never terminate:

```
def done(self, state):
    return False
```

- or terminate on state.count == 5

```
def done(self, state):
    (count, total) = state
    return count == 5
```

↑ returns true if count == 5

(Go cute how they do it in 1 line

I would do

```
if count == 5:
    return true
else:
    return False
```

Repeat

- repeats n times
- if n = None it repeats forever
- state = ~~#Pass~~ i, n
- whole machine must be done when leave it in done state

Sequence

- run one till done, start another, run till done, etc
- similar to repeat

Repeat Until and Until

- until a certain Boolean = True
- ↑ condition evaluated when TSM is done
- but sometimes we want to terminate as soon as it is true, at whatever step
  - so use general purpose Until combinator
  - it executes the machine
  - but only once

Week 3 HW

9/25

Now regular tutor HW

Cascading Machines

Need to read course notes + review lecture

9/26

Cascading Machine

- Write it as described in notes
- get next Values + start State
  - does not actually change state
- So this is 2 delay machines?
- well no, just superstructure

  foo1 = sm.Cascade(sm.____ , sm.____)

- init (s_1, s_2)
  - actual SMs → need to start up
  - Parallel in notes shows how
  - start state is a list of (sm1.startState, sm2.startState)
- same w/ state
  - So state can grow to multidimensional!

(2)

Well I want cascade, not parallel
- so don't need multidimensional state
- guess I do to keep track of both states
- what is difference here is that $O_1 = I_2$
"complete"
input and output are single values!

- try w/ 2 delay functions
  - works 1st try!
  - but I forgot how to use SMs
  - don't call directly getNextValue
  - but

```
foo = sm.Cascade( sm.Delay(1), sm.Delay(2))
foo.start()
Foo.step(3)
:
:
:
```

but errors in tutor -> O2 referenced before assignment
try to reproduce in IDLE
- get different error
- oh nvm I do

  oh when say sm._____ function it goes to SM lib

(3) So it was using their Cascade machine!!!

- and in 2nd machine input write O_1 not O_2!

now no errors and correct

- so I screwed up a little and bypassed my code!

- Passed tutor

Map

- passing functions as arguments
- no state machines

Part 1 mapList

- 2 arguments: procedure, list
- ~~Procedure~~
- returns list by applying procedure
- Must use list comprehension

↑ I was going to try anyway

easy!

```
return [f(x) for x in l]
```

nice + simple

passed tutor!

(4)

Part 2 sumAbs

- use mapList function
- w/ procedure sumAbs
- use built in sum + abs
- returns sum of the abs value of the #s
  - ? any storage, how to do it? ?
- Yeah must have sum storage
  - well accumulator
  - accumulator + abs(inp) = output
    = new accumulator
- but does not ~~use~~ use sum
  - must be accumulator?
- but how to store the accumulator?
- can't do w/ basic map list
  - only have access to current #
- Oh can it reference l from super class?
  - nope

~~ok~~ so try tutor to see test case
Oh!! they call sumAbs on (list)
- and that must use map List

not mapList(sumAbs...)
~~Soln~~

5

Makes far more sense now!

Done

Part 3 map Savare

- takes 2 arguments: Procedure w/ 2 arguments and a list
- so this multidimension list comprehension
- [f(x,y) for x in l for y in l]
- but that does [1, 2, 3, 2, 4, 6, 3, 6, 9]

  want [[1,2,3], [2,4,6], [3,6,9]]
- need 2 list comprehensions?
- or [[f(x,y), f(x,y)] for ...]
- well [[f(x,y) for x in l] for y in l]

- tutor: 2/3 correct
- wtf, difference does not work!
- fixed if I do f(y,x) for some reason

~~Don't HW!~~ Done

#3 Function Machines

- When making SMs want a pure function machine that can be used in a cascade
- does not have state
- define this pure function building block

- got it in 2nd try

```
init (self, f)
    self.f = f

get next value (self, state, inp):
    return state, self.f(inp)
```

Done HW!

Could do optional → Prank

- Parallel
  - use built in
  - it works w/ 2 outputs
  - now need to return the max of the 2
  - iteration over non seq
  - do I even need the pure function wrapper?
  - Oh just ignore error?
  - but its not doing max

6.01: Introduction to EECS I

Signals and Systems

Week 4 *September 28, 2010*

2nd of 4 sections

Outline

- Signals and systems view
- Operations on signals
- Feedback systems

Reading: Sections 5.1 – 5.4

Controlling car

Understanding systems

Good news:

- Have seen how state machines can control system's response to input
- Can build complex state machines to control our robots.

Bad news:

- Can't predict how our controllers are going to work, except by running them, possibly several times, and gathering data.
- When they don't work well, we don't have any systematic way of changing them to make them work better.

Solution:

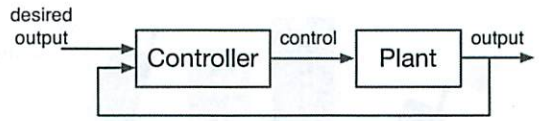
- Make **models** of the controller and of the robot and its world.
- Analyze the models mathematically to characterize performance and understand how to improve it.

relevant aspects

Prove controller ~~statemachine~~ will give you right ans

State machines as models

cascade w/ feedback



model of the world

- Make a state machine model of the **plant**: that is, the aspects of the external world that you are trying to control
- Make a state machine model of your **controller**
- Connect the state machines (cascade and feedback)
- Run it to see what happens

Computer programs are unpredictable

Could we figure out what will happen without running the simulation, just by looking at the definitions of the controller and the plant?

would be great!

In general, no.

It is **impossible to predict** even whether a general computer program will always terminate and produce a result.

- Known as the "Halting Problem" – very important result in computation.

Can show that it is not possible in general

LTI systems are predictable

Consider simpler class of state machines:

- **State:** last j inputs to the system, plus last k outputs of the system
- **Output:** a fixed linear function of the input and the state

Linear time-invariant (LTI) systems:

- Can be analyzed mathematically, to predict behavior without simulation
- Are compositional: cascade, parallel, and feedback compositions of LTI systems yield LTI systems

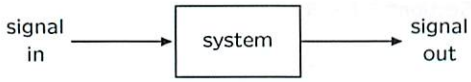
are predictable

but restrictive

-not so bad, good trade off

The signals and systems abstraction

Describe a system by the way it transforms inputs into outputs.

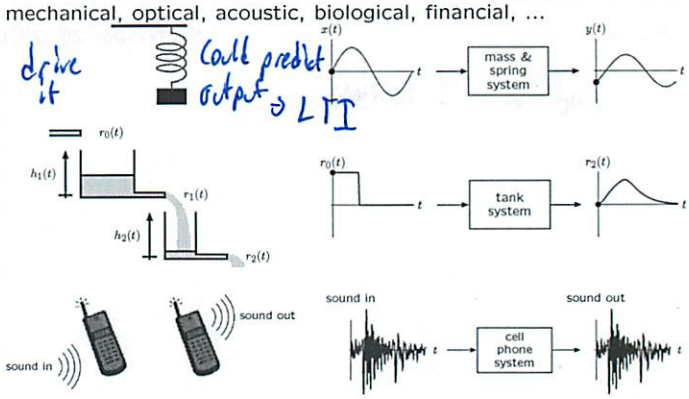


	Signals	Systems
Primitives:	unit sample, sinusoids	adder, gain, delay
Combination:	delay, scale, add	cascade, feedback
Abstraction:	signal	system function

Signals and systems: widely applicable

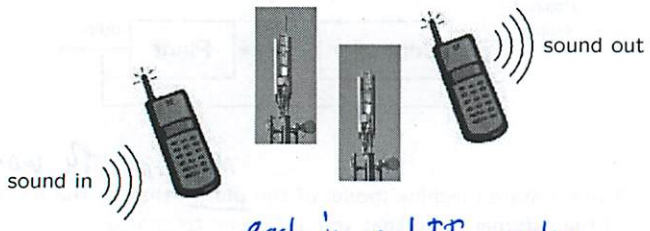
Signals and systems abstraction has broad application: electrical, mechanical, optical, acoustic, biological, financial, ...

drive it Could predict output ⇒ LTI

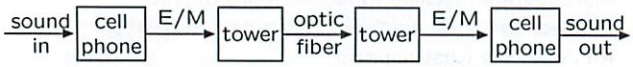


Signals and systems: modular

The uniform representation allows us to combine models of processes that operate upon different physical substrates.



each is a LTI system

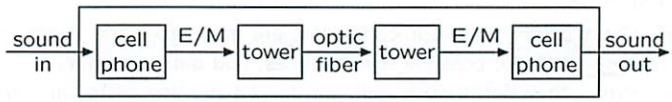


focuses on the flow of **information**, abstracts away everything else

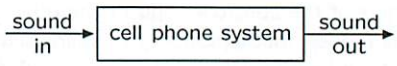
Signals and systems: hierarchical

Primitive representations can be combined into new representations.

Composition of **cascaded** systems:
the output of one is the input to the next

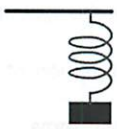


The new representation can be treated just like primitives.

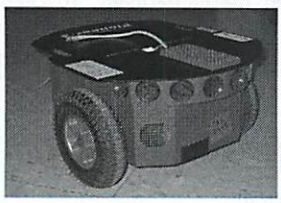


Continuous and discrete time

Inputs and outputs of systems can be functions of continuous time



or discrete time.



2, 1' sec I think

all decisions synced to clock

We will focus on discrete-time signals and systems.

Linear time-invariant systems

- **linear**: dependence of output on inputs is linear
- **time-invariant**: the same relationship between inputs and outputs holds for any value of n (i.e. for any instant in time)
- **causal**: sample response at time n only depends on values at the same or previous time steps

does not depend on time

can't look in future

Any LTI system can be described using a difference equation:

$$y[n] = c_0\, y[n-1] + c_1\, y[n-2] + \ldots + c_{k-1}\, y[n-k] + d_0\, x[n] + d_1\, x[n-1] + \ldots + d_j\, x[n-j]$$

Output $y[n]$ is a linear combination of

- k previous output values, $y[n-1], \ldots, y[n-k]$,
- j previous input values, $x[n-1], \ldots, x[n-j]$, and
- current input, $x[n]$.

Feed-forward systems

Difference equation defines the output of a system at a particular time point in terms **only** of its previous inputs.

Example:

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

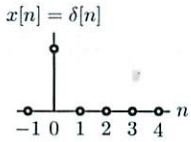
Step-by-Step solutions

Difference equations are convenient for step-by-step analysis.

Let $x[n]$ equal the "unit sample" signal $\delta[n]$,

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

↑ delta



This is our first example of a "primitive" (building block) signal.

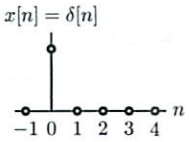
Step-by-Step solutions

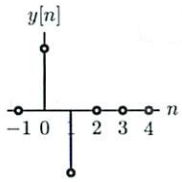
Difference equations are great for step-by-step analysis. Let

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

Using the diff eq:

$$\begin{aligned} y[n] &= x[n] - x[n-1] \\ y[0] &= x[0] - x[-1] = 1 - 0 = 1 \\ y[1] &= x[1] - x[0] = 0 - 1 = -1 \\ y[2] &= x[2] - x[1] = 0 - 0 = 0 \\ y[3] &= x[3] - x[2] = 0 - 0 = 0 \\ &\cdots \end{aligned}$$





discrete differentiator

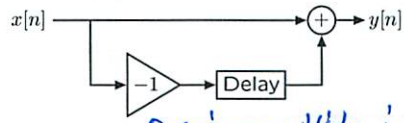
Block diagrams

Block diagram is alternative representation – captures interactions of components in graphical way

Difference equation:

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

Block diagram (delay, adder, gain):



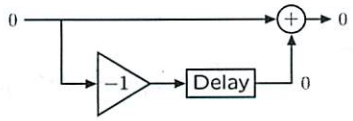
↑ gain → multiply input by

Same input-output behavior, different strengths/weaknesses:

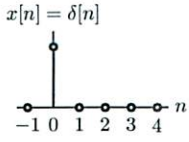
- **difference equations** are mathematically compact.
- **block diagrams** illustrate signal flow paths.

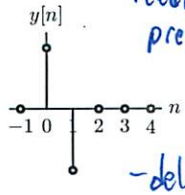
Block diagrams: Step-by-step solutions

Using the block diagram. Start "at rest." ← all 0



note not feedback of previous answer





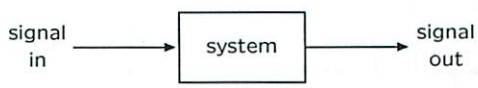
- delay is something that I never thought before

Signals

A **signal** is an infinite sequence of **sample** values at discrete time steps.

Common notational conventions:

- X: the whole input signal
- $x[n]$: the value of signal X at time step n
- Y: the whole output signal
- $y[n]$: the value of signal Y at time step n

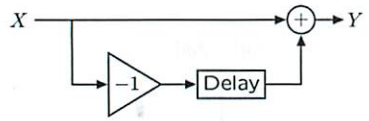


Systems *transduce* input signals into output signals.

Operations on signals

Operators manipulate signals rather than individual samples.

functionals



Wires represent whole signals (e.g., X and Y).
The boxes **operate** on those signals

Unit sample signal

Only crucial **primitive** in our PCAP system:

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

Other useful primitives are **step** and **sinusoid** signals. Discussed in software lab and readings.

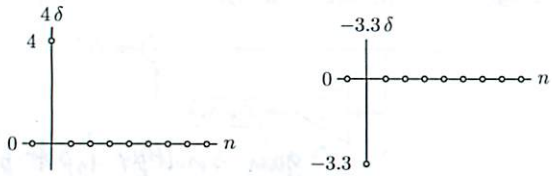
building up PCAP

Operations on signals: scaling / gain

Multiply a signal X by a constant c:

operation on signal

$$(c \cdot X)[n] = c \cdot x[n]$$



Constant c often called a **gain**.

(for historical reason just multiply each value by

Operations on signals: delay

Shift signal X to the right (forward in history), getting $\mathcal{R}X$ (Think of this as delaying the input signal so that it shows up one or more instances of time later):

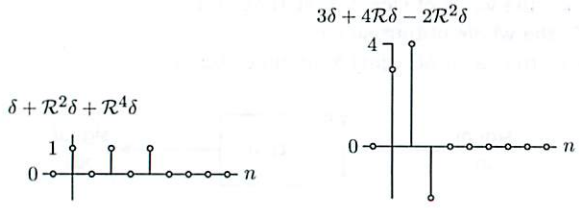
R for right

$$(\mathcal{R}X)[n] = x[n-1]$$

Operations on signals: addition

Add signals X_1 and X_2 together to get a new signal $X_1 + X_2$:

$$(X_1 + X_2)[n] = x_1[n] + x_2[n]$$



↑ can build any signal w/ δ - complex though

Abstracting signals

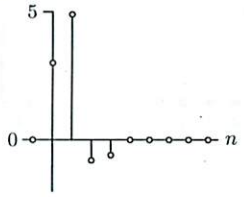
Scaling, delay, addition all return new signals that can be further combined

Abstract by naming

$$Y = 3\delta + 4\mathcal{R}\delta - 2\mathcal{R}^2\delta$$

then operate on Y

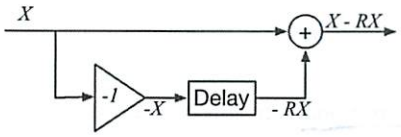
$$Z = Y + 0.3\mathcal{R}Y$$



Any signal with finitely many non-zero samples can be constructed from δ with delay, adder, and gain operations.

Operator notation

Representing the difference machine



with $\mathcal{R}$ leads to the equivalent representation

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

Operator notation: Check yourself

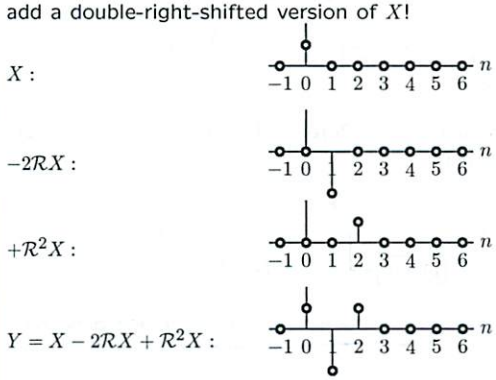
Let $Y = \mathcal{R}X$. Which of the following is/are true:

1. $y[n] = x[n]$ for all n
2. $y[n+1] = x[n]$ for all n $= Y[n] = X[n-1]$
3. $y[n] = x[n+1]$ for all n
4. $y[n-1] = x[n]$ for all n
5. none of the above

Y = output

Operator algebra

Operator notation prescribes operations on signals, not samples: e.g., start with X, subtract 2 times a right-shifted version of X, and add a double-right-shifted version of X!

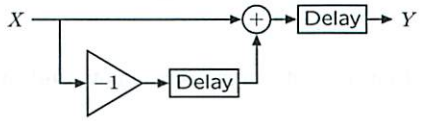


Operator algebra: commutativity

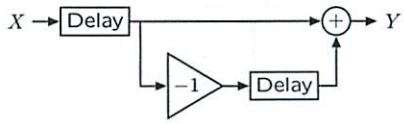
Expressions involving $\mathcal{R}$ obey many familiar laws of algebra, e.g., commutativity. $X \cdot Y = Y \cdot X$

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

This is easily proved by the definition of $\mathcal{R}$, and it implies that cascaded systems commute (assuming initial rest)



is equivalent to



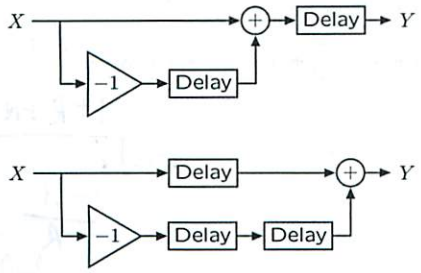
Operator algebra: distributivity

Multiplication distributes over addition.

Equivalent operator expressions:

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

Equivalent systems



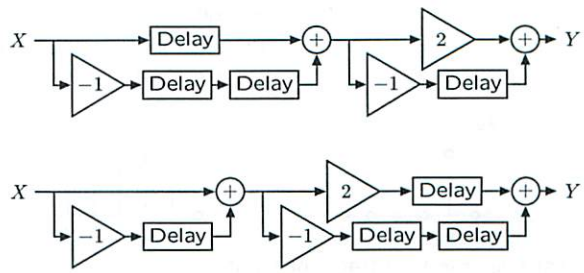
Operator algebra: associativity

The associative property similarly holds for operator expressions.

Equivalent operator expressions:

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

Equivalent systems



treat just like algebra

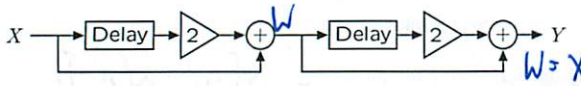
What are the properties of the system?

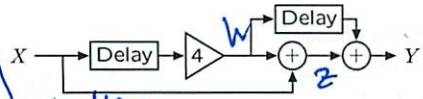
Check Yourself

- not the signal now

analize each piece -> PCAP

How many of the following systems are equivalent?





$W = X + RX$
$= (R+1)X$
$Y = X + 4RW$
$= X + 4R(R+1)X$
$= (4R^2+4R+1)X$

$W = 4RX$
$Z = W + X$
$Y = Z + RW$
$\hookrightarrow = (4R^2+4R+1)X$

$W = X + 2RX$
$= (2R+1)X$
$Y = W + 2RW$
$= (2R+1)W$
$= (2R+1)^2 X$
$= (4R^2+4R+1)X$

System Functional

The system is completely characterized by the relationship between the input signal X and the output signal Y.

We call this relationship,

$$\frac{Y}{X}$$

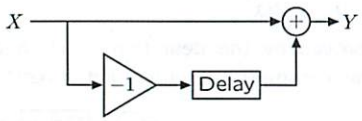
the **system functional**. It is independent of any particular input signal.

So far, the system functionals that we have seen are polynomials. More generally, they will be ratios of polynomials.

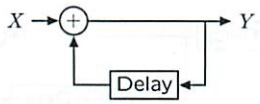
can say things about the system * for any X

Feedforward and feedback systems

Feedforward: output depends only on previous inputs

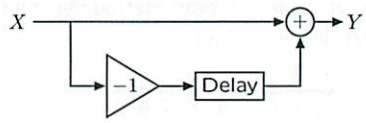


Feedback: output depends on previous inputs **and outputs**



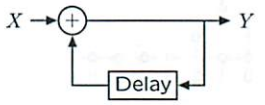
Explicit and implicit rules

Recipes versus constraints.



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

Recipe: output signal equals difference between input signal and right-shifted input signal.

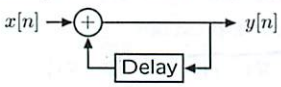


$$Y = \mathcal{R}Y + X$$
$$(1 - \mathcal{R})\,Y = X$$

Constraints: find the signal Y such that the difference between Y and $\mathcal{R}Y$ is X. But how?

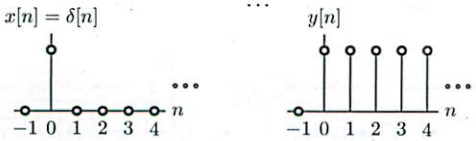
Example: accumulator / integrator

Step-by-step analysis always works. Start "at rest."



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

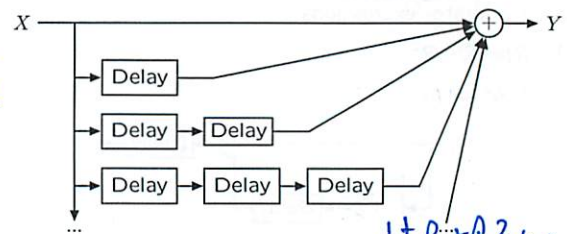
$$y[0] = x[0] + y[-1] = 1 + 0 = 1$$
$$y[1] = x[1] + y[0] = 0 + 1 = 1$$
$$y[2] = x[2] + y[1] = 0 + 1 = 1$$
$$\cdots$$



Persistent response to a transient input!

Example: accumulator

An equivalent system:



$$Y = (1 + \mathcal{R} + \mathcal{R}^2 + \mathcal{R}^3 + \cdots)\,X$$

is equivalent to

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

Proof in readings.

$\frac{Y}{X} \to \frac{1}{1-R}$

$1 + R + R^2 + \cdots$ divided by $1 - R$ into 1

$\frac{1-R}{R}$

$\frac{R - R^2}{R^2}$

...

can still manipulate as though polynomials

Linear time-invariant systems

Any LTI system can be described using a difference equation of the form:

$$y[n] = c_0\, y[n-1] + c_1\, y[n-2] + \ldots + c_{k-1}\, y[n-k] + d_0\, x[n] + d_1\, x[n-1] + \ldots + d_j\, x[n-j]$$

State is

- k previous output values and
- j previous input values.

Output $y[n]$ is a linear combination of:

- k previous output values, $y[n-1], \ldots, y[n-k]$,
- j previous input values, $x[n-1], \ldots, x[n-j]$, and
- current input, $x[n]$.

Examples

$$y[n] = c_0\, y[n-1] + c_1\, y[n-2] + \ldots + c_{k-1}\, y[n-k] + d_0\, x[n] + d_1\, x[n-1] + \ldots + d_j\, x[n-j]$$

- Output at step n is 3 times the input at step n:

  $y[n] = 3x[n]$

  dCoeffs: 3, cCoeffs: none
- Output at step n is the input at step $n-1$:

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

  dCoeffs: 0 1, cCoeffs: none
- Output at step n is 2 times the input at step $n-2$:

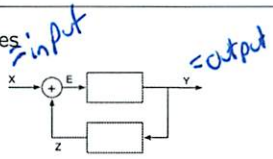
  $y[n] = 2x[n-2]$

  dCoeffs: 0 0 2, cCoeffs: none

built algebra for operators

Combining modules

Assign names to all wires



input output

Write operator equation relating input(s) and output of each component

$$Y = kE$$

$$E = X + Z$$

$$Z = \mathcal{R}Y - \mathcal{R}^2Y$$

Solve for output

$$Y = kX + k\mathcal{R}Y - k\mathcal{R}^2Y$$

could pull out ratio of polynomials

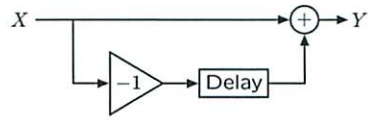
Convert to difference equation

$$y[n] = kx[n] + ky[n-1] - ky[n-2]$$

way you can go back + forward b/w

LTI systems as state machines

model it




```
class Diff(SM):
    startState = 0
    def getNextValues(self, state, inp):
        return (inp, inp - state)
Diff().transduce([1, 0, 0, 0])
```

input signal

Or

& that's why they called it that

```
diff = sm.ParallelAdd(sm.Wire(),
                      sm.Cascade(sm.Gain(-1), sm.R(0)))
diff.transduce([1, 0, 0, 0])
```

Delay

Multiple representations of LTI systems

- **difference equations**: good for step-by-step simulation
- **block diagrams**: good for signal-flow intuition
- **operator expressions**: good for compact description and combining systems
- **Python SM subclasses**: implementation of difference equation
- **Python combination of primitive SMs**: implementation of difference equation, but easier to get right

This Week

Software lab: Class for manipulating signals; can do same operations as a feed-forward system

Design lab: Constructing and using LTI model for robot control

To get help:

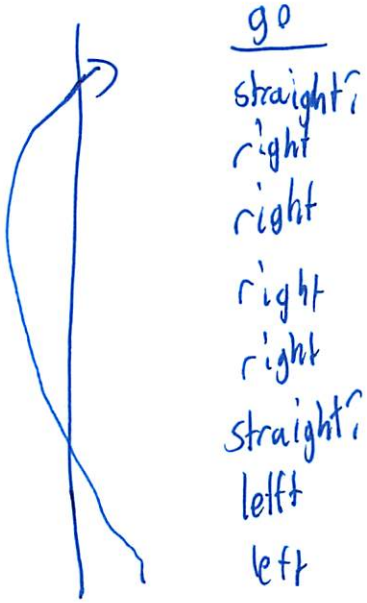
- Email `6.01-help@mit.edu`
- Go to lab hours (see course web page for times)
- Remember to check your due dates/times on the tutor
- Interviews will be held on October 17 and 18; and on November 14 and 15

Week 4 Lecture

9/28

Controlling a system

- Say driving car
- sensor to sense your position in a lane
- then correct if wrong



go
straight
right
right
right
right
straight
left
left

but overstear

Using LTI

- combine primitives to abstract problem
- analize
- ...
- ...

Abstracting Signals

- unit ~~step~~ sample is only basis signal we need
- unit step - goes to unit sample - so also basis signal
- can create any ~~signals~~ signals with this

(2)

Commutative

$R(1-R)x$

$R(1-R)x[n]$

$R(x[n]-x[n-1])$

$x[n-1]-x[n-2]$

$(1-R)Rx$

$(1-R)Rx[n]$

$(1-R)x[n-1]$

$x[n-1]-x[n-2]$

Algebra

- operators satisfy algebra
- have property of closures

Reference

Thoughts

9/28

HS prepared me bad for math

but I can learn it!

- if it visualized → best case
- like P-Set yesterday
- just need to go slow + understand
- which I've been doing
  - but not struggling
    - kinda miss that from last year

Really like when stuff applies to real world

- CS
- Prob
- Then can visualize
- and my interests

I learned in middle school

- don't get comfortable
- can always get better

Software Lab 4: Signal Class

1 Setup

- **Using a lab laptop or desktop machine**: Log in using your Athena user name and password; in the terminal window, type `athrun 6.01 update`. Work in the directory `Desktop/6.01/lab4/swLab`.
- **Using your own laptop**: Download the zip file for software lab 4 from the *Calendar* tab of the course web page, unzip it, and work in the resulting directory.

Starting Idle so you can make plots:
If you are using your own laptop, be sure you have Numpy installed. (See the software tab of the course web page for instructions on how to do it. Note that Numpy doesn't work with 64-bit Python on Windows.)

- If you are using **a MacOS, Linux, lab laptop or desktop machine**: type `idle -n` in a terminal window.
- If you are using a **Windows machine**: download and unzip the `idle-n.bat` file under Week 4 on the Calendar page. Then, to start idle, run the `idle-n.bat` file.

If you have trouble doing this on your computer, ask a staff member right away.

2 Introduction

The software lab for this week is to implement a Python class that provides methods for performing operations on signals. You can think of any signal as an infinite sequence of sample values. We will represent signals using a function that can be called with any time index, returning the value of the signal at that index.

We will use object-oriented software techniques to organize our software for implementing and manipulating signals. An abstract superclass, called `Signal`, will have methods for performing operations on signals, no matter how they are represented. The `Signal` class will only assume that every instance has a method called `sample`, which takes a time index, `n`, and returns the value of the signal at time `n`. Then, we will have several subclasses of `Signal` whose only job is to implement the `sample` method in an appropriate way.

The simplest signal we could imagine having is one with a constant value. Here is its class definition:

```
class ConstantSignal(Signal):
    def __init__(self, c):
        self.c = c
    def sample(self, n):
        return self.c
```

The `UnitSample` signal has value 1 at time 0 and value 0 at all other times:

```
class UnitSampleSignal(Signal):
    def sample(self, n):
        if n == 0:
            return 1
        else:
            return 0
```

Somewhat more interesting is a sinusoidal signal; we implement this with a `CosineSignal` class, whose constructor takes a frequency `omega` and a phase `phase`:

```
class CosineSignal(Signal):
    def __init__(self, omega = 1, phase = 0):
        self.omega = omega
        self.phase = phase
    def sample(self, n):
        return math.cos(self.omega * n + self.phase)
```

A method `plot(self, start, end, newWindow, color)` is also defined for all signals. It plots the signal from `start` to `end`. If `newWindow` is a string, it will make a new window for this plot and give it that title, otherwise if `newWindow` is `False` it will plot this signal in the currently working window; `color` is a string specifying the color of the new curve to be added to the plot. The parameters `start`, `end`, `newWindow` and `color` all have reasonable default arguments, so you can just specify the ones you need.

Plotting will *only* work from `idle` if you start it with the `-n` flag. Currently, the first signal plotted in a new window sets its size; plot the signal with the largest range first.

3 Play with Signals

Step 1.

- Open the file `sl4Work.py` in Idle. It imports our implementation of the module `sig`, which defines all of the classes and methods described in this handout. So, you can use those classes and methods in calls that you make in this file. For instance, to make a unit sample signal, you can do:

  ```
  s = UnitSampleSignal()
  ```

- Add a Python command in that file that will make a plot of the unit sample signal. If `s` is a signal, then

`s.plot(-5, 5)`
will plot that signal in a new window, with samples from time steps -5 to 5.

- Choose Run Module in Idle (under the Run menu in the window with the `sl4Work.py` file). You should see the plot. Be sure it makes sense to you.
- Now make a cosine signal and plot it, in the same way. Try different values of `omega` and `phase`.

4 New Kinds of Signals

This section describes several new subclasses of `Signal`. In this lab, you will start by doing an exercise using our implementation of them, and once you have some experience with using these signal classes, we will ask you to implement your own version of them.

- `StepSignal()`: has value 0 at any time $n < 0$ and value 1 otherwise.
- `SummedSignal(s1, s2)`: takes two signals, `s1` and `s2`, at initialization time and creates a new signal that is the sum of those signals. Note that this class should be *lazy*: when the signal is created, no addition should happen; values only need to be added when a sample of the summed signal is requested.
- `ScaledSignal(s, c)`: takes a signal `s` and a constant `c` at initialization time and creates a new signal whose sample values are the sample values of the original signal multiplied by `c`.
- `R(s)`: takes a signal `s` at initialization time and creates a new `Signal` whose samples are delayed by one time step; that is, the value at time n of the new signal should be the value at time $n - 1$ of the old signal.
- `Rn(s, n)`: takes a signal `s` at initialization time and creates a new `Signal` whose samples are delayed by `n` steps.

Polynomials in R

We have built up an algebra of signals, which is very general. We can add, scale, and delay signals. One way to express combinations of these operations on a signal is to describe them in terms of a polynomial in $\mathcal{R}$. So, we will implement a procedure to construct a new signal by operating on it with a combination of sums, scales, and delays described by a polynomial in $\mathcal{R}$.

So, if X is a signal, then we can describe a delayed version of X as $\mathcal{R}X$, and a version of X that has been delayed by two time steps as $\mathcal{R}\mathcal{R}X$. That same signal, scaled by 7, would be $7\mathcal{R}\mathcal{R}X$ or $7\mathcal{R}^2X$. If we added $7\mathcal{R}^2X$ to X, then we'd have $7\mathcal{R}^2X + X$, which we can think of as the signal X transformed by a polynomial with coefficients `[7, 0, 1]`. So, we could define a procedure

- `polyR(s, p)`: takes an instance of the `Signal` class (or one of its subclasses), `s`, and an instance of the `Polynomial` class, `p`, and returns a new signal that is `s` transformed by a polynomial in $\mathcal{R}$.

5 Constructing Signals

First, we will use an existing implementation of the signal constructors to build complex signals from primitive ones, in order for you to get an intuition about how they work.

Step 2. To do tutor problem Wk.4.1.1, do the following steps:

- Go to the 6.01 web page, look under the *Reference Material* tab, and click on *Software Documentation*. If you look at the documentation for the module `sig`, you can see the various methods and functions discussed above, which are defined in the `sig` module.
- Edit the file `sl4Work.py` in Idle to construct new instances of signals by making combinations (e.g., sums, scalings, and delays) of instances of the basic `UnitSampleSignal` and `StepSignal` classes, as specified in the tutor problem. **You can use any of the classes or procedures in the `sig` module. You should not write any new Python classes for this problem!!**
- Plot them to be sure they're right.
- Remember that the `polyR` class takes as a second argument an instance of the `Polynomial` class. You can create new instances of this class via `poly.Polynomial(c)` where `c` is a list of coefficients.

Wk.4.1.1 Do tutor problem Wk.4.1.1.

6 Implementing Signals

Now, we will implement the classes we just used to make complex signals out of simple ones. Our implementations should have the same behavior as what you experienced in the previous section.

Step 3. The remaining problems ask you to implement each of the new subclasses of `Signal`, as described above. Each one only needs to supply an appropriate `sample` method, and possibly an `__init__` method. We have provided you with a file, `sl4Skeleton.py`, that you can use to debug your implementations before entering them in the Tutor. You should enter your new class definitions at the end of `sl4Skeleton.py`. You can test them by putting your test cases in `sl4SkeletonWork.py`, which imports `sl4Skeleton.py`, and using Run Module on `sl4SkeletonWork.py` in Idle.

You can only use the methods and classes that are defined in `sl4Skeleton.py`; you will extend that file to be a complete implementation of the `sig` module.

It is very important that, for testing, you use `sl4SkeletonWork.py` which imports your class definitions, rather than `sl4Work.py`, which imports our class definitions.

Check Yourself 1. Whenever you define a new signal class, make an instance or two and plot them to be sure you're getting something reasonable.

Wk.4.1.2-4 Do tutor problems Wk.4.1.2 through Wk.4.1.4

Check Yourself 2. We have defined `usamp` to be an instance of the `UnitSample` class. What does the signal `polyR(usamp, poly.Polynomial([3, 5, -1, 0, 0, 3, -2])` look like?
Do you see how you could use `polyR` to construct any signal (with finitely many non-zero values) out of the unit sample?

Step 4. Implement the **procedure** `polyR(s, p)`, which takes an instance of the `Signal` class (or one of its subclasses), `s`, and an instance of the `Polynomial` class, `p`, and returns a new signal that is `s` transformed by a polynomial in $\mathcal{R}$. So, for example, if `p` is an instance of the `Polynomial` class, with `p.coeffs = [3, 2, 10]`, then the result of `polyR(s, p)` is a signal

$$3\mathcal{R}\mathcal{R}s + 2\mathcal{R}s + 10s$$

In your implementation, you can use any of the previously implemented signal constructors, including `Rn`. Note that you don't need to define a class to do this: just writing a single procedure will suffice. Depending on how you choose to write your code, you may find it helpful to remember that any list, such as `test`, can be reversed by calling `test.reverse()`

Wk.4.1.5 Do tutor problem Wk.4.1.5.

Sw Lab 4

9/28

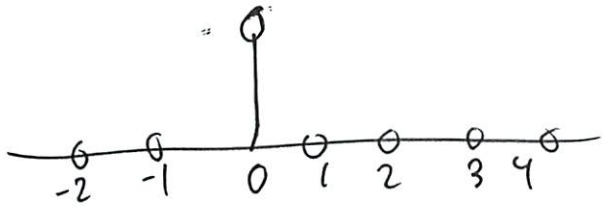
Need Numpy

performing operations on signals

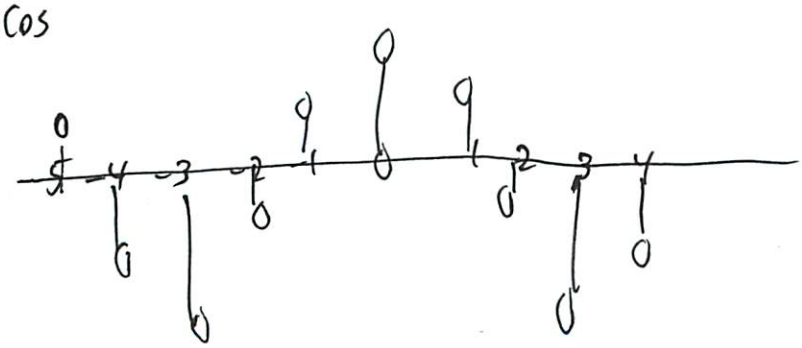
OO

- superclass signal
  - - sample method
    - returns signal at time n
  - must implement

- but is signal not a list [...]
  - no sample
    - n is current signal time
    - not signal[n] ∈ what I thought
- oh signal generator
- plot predefined
- cool



- try cos



- what are omega + phase in? - radians?

(2) new Subclass of signal

- bunch of them

algebra of signals

R = delay

RR = R^2 = 2 delays

7X = scale by 7

7RRX = $7R^2X$

+X → can add } notation not right here

$7R^2X + X$

→ is like polynomial [7, 0, 1]

poly R(s, P)

↑ ↑ polynomial class

signal

class ~~A~~
just did

- generates signal

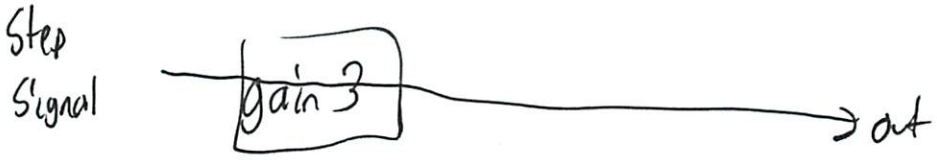
---

tutor 4.1.1 (generators)

- signal constructors predefined
- all classes we need are defined
- construct 4 signals
- testing will ~~return~~ be is signal in range
- poly R takes polynomial

(3)

- plot to make sure stuff is right
- ok get started
  - step 1
    - signal = 3 for $t \geq 3$
      0 otherwise
    - want us to modify old signal, right?
    - scale, shift, etc old signals



? is that it?

but how

Step1 = stepSignal()

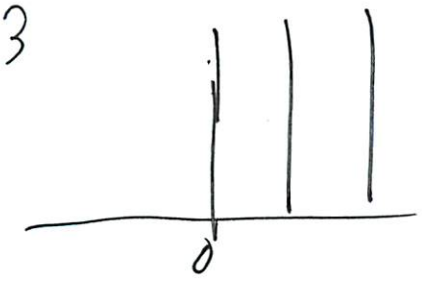
Step1. gain(3)

↑ not it → its a state machine

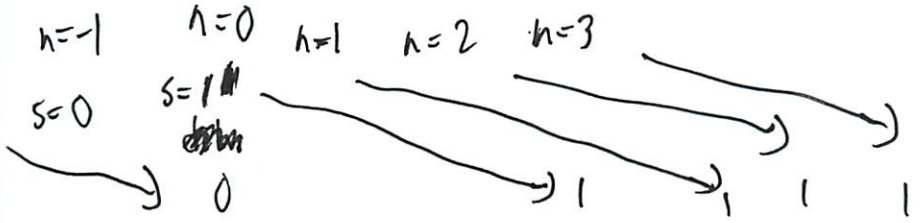
- do normal SM
- not using anything SM
- sm w/ summed signal
- gain use → Scaled Signal
  - use only those on sheet
  - ↑ use this
  - prob internally its a SM
  - not like lecture today

(4)

that was wrong



- ~~Read~~ try w/ delay
  - right delayN(step signal , 3)
- so what are we doing



- should = 3 - so scale signal (delayn (step signal(), 3), 3)
  ✓ Done

Step 2

~~the~~ scale signal(delay n (step signal (), 7) -3)

✓ Done

Step up down

= 3 for time $3 \leq t \leq 6$

= 0 otherwise

- Hint can you use step 1 + 2

(5)

; step 1 ≠ step 2

- can only add
- will add - of it to subtract
- already neg

Summed Signal (step 1, step 2)

- bingo
  - for 7 → ∞ 3 + -3 = 0

done ✓

- they set that up!
- how would I have done it regardless
- guess would have to do that
- just would not have thought of doing it like that

Step up down Poly

- use Poly & class
- + { 1 → 1.0
  3 → 3.0
  5 → 5.0
  otherwise → 0
- poly & predefined for us
- what do I want to feed in

scale (delayn (step, 5) 5) + scale (delayn (step, 3) 3) + ...

(6)

So x = step signal

$5R^5x + 3R^3x + 1Rx$

$[5, 0, 3, 0, 1, 0]$

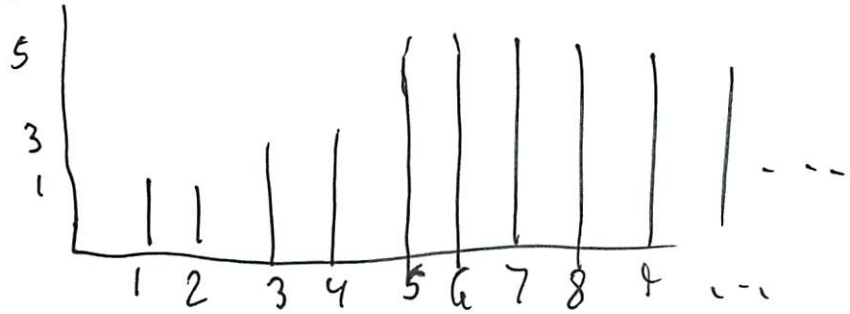
? ↑
R5 R4

- so polyR (stepC), $[5, 0, 3, 0, 1, 0]$)

well polynomial instance of that

how does it use step up down?

Ok ans



- Ok not want
- think negitive

$[5, -3, 3, -1, 1, 0]$

↑ but then what up front ~~# add to 2p~~

Scaled (Rn (step, 6, -5)

+ add to PolyR

(7)

bingo ✓ 1st modified try

- this is kinda fun

now time to check

- oh they used Unit Sample Signal
  - as a base
  - cleaner
  - but I like how I solved it ☺

4.1.2

Implement signals

- to make more complex signals
- as Signal subclass
- need sample() method and perhaps __init__
- define in Skeleton.py ⟵ imports
- test in SkeletonTest.py
- test each time by plotting

---

has value 0 at time $n < 0$

1 otherwise

$$\begin{cases} 0 & \text{if } n < 0 \\ 1 & \text{otherwise} \end{cases}$$

(8)

- So signal () method
  - Convert UnitSample signal
- bingo lst tru ✓

Summed + Scaled signal

- implement sample()

Sum

return s1.n + s2.n
(self. self.)

Or s1[n]

- need some accessor method
  ~~step~~. sample! sample(n)

- oh depends which you run from in IDLE when multiple windows open → run in Skeleton Work [F5]
- works self.s1.sample(n) + self.s2.sample(n)

Scaled

self.s.sample(n) * C
(self.)

↑ stop forgetting

check in tutor
✓ Got it

(9) Additional Subclasses: R and Rn

- ha I was wondering how they implemented Rn
- now I have to do it!

- R

  - need to store somehow
  - sum?
  - or sample(n-1)
  - yeah thats it

Rn

Sample(n-k)
↑self. of course

✓ Bingo

4.1.5 Poly R on Signals

- _add_ and _Rmul_ are there
  - ~~so I can scale~~
  + only works on 2 signals
  - need to specify start value if use python's sum() built in function
  - first arg: signal
    2nd: polynomial
  - I think it is time to look up my polynomial class
  - Can use polynomial class to do work

at home now

So what do I want to do

- multiply signal by polynomial

$$[5,0,3,0,1,0] = 5R^5x + 3R^3 + 1Rx$$

- returns a signal

- can use Rn

~~- for each~~

flip polynomial

for each # Rn by # position

Scaled Multiply for value

- try revising code

Completely not in mind set for this

need to install numpy

- not working!
- could go to cluster
- or VM
- tried installing from web
- now it works, guess you had to do that

- oh - using their poly lib

- I think this will be a few steps

(11)

Is there a method for finding polynomial lenght?

how find max

- or start from bottom

Produced something

- is it right?
- guessed at max lenght

- check us their script
  - hope - not even close!
- Ok works for 1
- it worked when I took out the weird code
- yeah I think thats it
  - try in tutor

I don't think its anything like what they were thinking

- oh well

Tutor major fail

- they don't have Polynomial.coeff when ~~test~~ 6.01 lib docs

prob not supposed to do it that way

- sent email to 6.01-help

(12)

How else would I do it though
- no method to get polynomial out as a list
- they are prob thinking polynomial add, mul
  - but how to do that w/ abstraction?

Looked up G.O! documentation
- is coeffs variable
- but no coeff() method
- even better!

And I remembered need to work w/ input!
- list starts at front → need to reverse
- nice works
- but need to take input signal!
  - s.step at that value n ??
  - or just s
    - we want signal
- cool same ans as ~~office~~ theirs - try

Woot!! I figured it out!
- now just need pre lab

Pre Lab

9/30

difference equation math

$$y[n] = c_0 y[n-1] + c_1 y[n-2] + \cdots + c_{k-1} y[n-k] + d_0 x[n] + d_1 x[n-1] + \cdots + d_j x[n-j]$$

- Specify d Coeffs and C Coeffs
  x = input
  y = output

Enter a sequence of #s

Output at time n = $\sum$ inputs including n

- So what to do ???

  input 1, 1, 1, 1

  but how many?

  none none?

No + no

is it in the reading?

like last lecture

---

k = previous output

j = previous input

linear combo of k previous outs + j previous ins + current x[n]

(2)

Seems stupid

there are no sum examples in lecture

nothing in course notes

Web seach

- nothing is summing over

examples

- can't find a better

- output = $3x[n]$

$d = 3$ c = none

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

$d = 0$ 1 none

$y[n] = 2x[n-2]$

0 0 2 none

$y[n] = 2y[n-1]$

none 2

$y[n] = y[n-1] + y[n-2]$

none 1 1

* kinda see

- but what is sum of past values

none 1 1 1 1 -- how far?

(3)

Why do they provide no help??

Emailed in

- but no response to my email from before
- ? do ~~that~~ morning in half hr break?
- no other way forward
- or let it expired

and I don't want to miss nano quiz

---

ah found other course notes

- no help

One more in old lecture

$$y[n] = x[n-1] + y[n-2]$$

$d = 0\ 1$ $\quad c = 0\ 1$

held to review whole thinking

- tomorrows lab!

Oh just for my problem

none 1

1 1

current input, past output

bingo!

(4)

ok can I get rest

Output at n = sum of inputs up to n-1

0 | |

? previous input | output

✓ bingo

Output at n = scaled input up to n-1

~~0 1~~

0 .1 | 1

✓ done!

So glad figured it out

rand 4

9/30

Construct a signal

tictoclk

Oh! use existing signals

- not define

---

n = position

PolyR [0, -1, 0, 2]

✓ Done

- had lots of name space issues

Confused on Cosine one

- how to do?
- failed
- no idea how to do
  - so got 1/2 done

Design Lab 4: Hitting a Wall

1 Introduction

This lab should be done with a partner. Each partnership should have a lab laptop or a personal laptop that reliably runs soar. Do `athrun 6.01 update` to get the files for this lab, which will be in `Desktop/6.01/lab4/designLab/`, or get the software distribution from the course web page.

The relevant files in the distribution are:

- `dl4Work.py`: A file with appropriate imports for making state machines and plotting their outputs.
- `wallFinderWorld.py`: A world with a wall for the robot to approach.
- `smBrainPlotDistSkeleton.py`: A simple brain with a place for you to write the state machine for the controller.

Be sure to mail all of your code and plots to your partner. You will both need to bring it with you to your first interview.

In Design Lab 3, you developed a "distance-keeping" brain to position a robot a fixed distance in front of a wall (even when the wall moved). This week, we will develop a model of that system, and we will use the model to understand the performance of the brain/robot system (Were the responses fast or slow? Did they overshoot or oscillate?).

The brain/robot system is an example of a simple control system with a single input (the desired distance to the wall) and a single output (the actual distance to the wall). We can think of the control system as having three subsystems: a controller (the brain), a plant (the robot's locomotion system), and a sensor (the sonars).

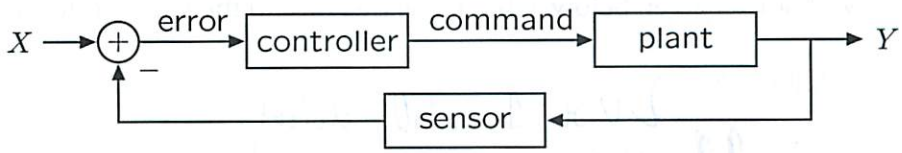


Figure 1 Structure of Control System

Think of the input as being set by a user (e.g. you) or by some planning system that the robot uses to navigate in the world (e.g. a state machine that moves the robot from one target point to the next in some sequence). Thus, the input might stay constant for some period of time, and then change to a new value for some period of time, and so on.

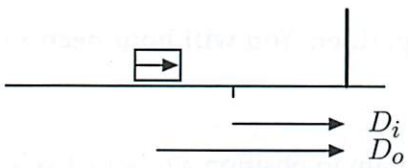
Generally, we design the controller to *command* the plant so that its (the plant's) output Y tracks some desired value X. For example, in the case of asking a robot to stay a particular distance from

a wall, we want the output Y to be as close as possible to the input X. Since the plant is typically a physical system, the output of the plant (e.g., the position of the robot) is not easily compared directly with X. Rather, the physical output is measured by the sensor, whose output (which is typically electronic) can be subtracted from X to determine the *error*. Ideally, the output of the sensor is proportional to the output of the plant. More generally, however, the sensor introduces its own distortions, delays, and noise.

Wk.4.2.1 **If you haven't already done it, do tutor problem Wk.4.2.1.**

2 Difference equations for wall finder

Make a simple model of the brain/robot system, as follows. Let D_o (the 'o' stands for *output*) represent the current distance from the robot to the wall, and let D_i (the 'i' stands for *input*) represent the desired distance to the wall. Also let V represent the forward velocity of the robot. Let $T = 0.1$ seconds represent the time between steps.



Assume that the robot immediately achieves the commanded velocity when it receives the command and that it maintains that velocity until it receives the next command. In other words, the robot is able to instantly achieve a particular velocity, and it holds that velocity constant until asked to change it.

Check Yourself 1. Assuming these velocity commands and the initial actual distance to the wall are as given below, what is the distance to the wall on step 1?

$V[0] = 1$ $D_o[0] = 3$

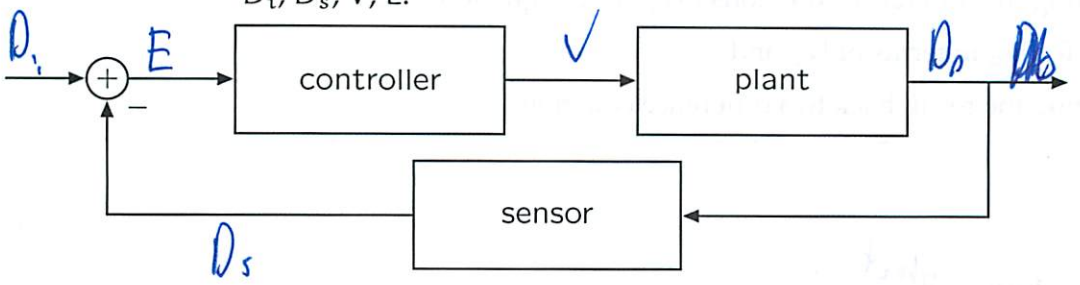
$V[1] = 2$ $D_o[1] =$ [illegible] 2V of 1 still active 2.9 since time is .1 faster, 1 sec latter

Assume the system has the structure shown in **Figure 1**. Assume that the sensor measures the current distance D_o and generates the sensed distance D_s, which is equal to the current distance D_o delayed by one step time. Let E represent the error signal $D_i - D_s$. On each step, the controller should command a forward velocity V in proportion to the error so that $V = kE$. Choose k so that the velocity is 5 m/s when the desired location is 1 m in front of the robot (think about the previous figure showing the position of the robot in order to help frame this calculation for k).

$5 = k1$

$k = 5$

Check Yourself 2. Fully label the following diagram for the brain/robot system. Include D_o, D_i, D_s, V, E.



Write down, using constants T and k as well as the signals you labeled on the figure, difference equations that relate the inputs and outputs of

- the controller $V = kE$
- the model of the plant $D_o = V[T] \cdot T + D_o$
- the model of the sensor $D_i - D_s$

$D_s = D_i - E$ $D_s = D_{i0} +$ ...

See sheet

Wk.4.3.1.1-3 Enter the coefficients of your difference equations into parts 1, 2, and 3 of tutor problem Wk.4.3.1.

E = delayed actual

See sheet → lots of work

Combine these equations to derive a difference equation that relates D_o to D_i, by:

1. Converting the difference equations to operator equations in $\mathcal{R}$,
2. Solving for D_o in terms of D_i, and
3. Converting the result back to a difference equation.

See sheet

Wk.4.3.1.4 Enter the coefficients of your difference equation into part 4 of tutor problem Wk.4.3.1.

3 State machines primitives and combinators

We can create all linear time-invariant systems by combining two primitive state machines in different ways: `sm.Gain` and `sm.R`.

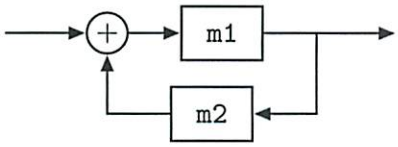
The `sm.Gain` state machine is really just a pure function: the output at step n is the input at step n, times a constant, `k`. The state is irrelevant. The reason we create this as a type of state machine is that we want to use the principles of PCAP to be able to combine it with other state machines to create new kinds of state machines.

```
class Gain(SM):
    def __init__(self, k):
        self.k = k
    def getNextValues(self, state, inp):
        return (state, self.k*inp)
```

The `sm.R` state machine is a renamed version of the `Delay` state machine. It takes a value at initialization time which specifies the initial output of the machine; thereafter, the output at step n is the input at step $n-1$.

```
class R(SM):
    def __init__(self, v0 = 0):
        self.startState = v0
    def getNextValues(self, state, inp):
        return (inp, state)
```

For the purposes of building LTI systems, the *feedback addition* composition will be useful. It takes two machines and connects them like this (note that we are using generic boxes here, those boxes would be a triangle if the state machine were simply a gain, or would be labeled with an R if the state machine were a delay, or could be some more complicated feedback loop):



If `m1` and `m2` are state machines, then you can create their feedback addition composition with

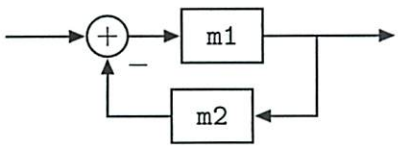
```
newM = sm.FeedbackAdd(m1, m2)
```

Now `newM` is itself a state machine. So, for example, `newM = sm.FeedbackAdd(sm.R(0), sm.Gain(1))` makes a machine whose output is the sum of the inputs from step 0 up to but not including the present step. You can test it by feeding it a sequence of inputs; in the example below, it is the numbers 0 through 9:

```
>>> newM.transduce(range(10))
[0, 0, 1, 3, 6, 10, 15, 21, 28, 36]
```

Feedback subtraction composition is the same, except the output of `m2` is *subtracted* from the input, to get the input to `m1`. (Note the minus sign next to the output of `m2` as it is fed into the adder.) You can use it like this:

```
newM = sm.FeedbackSubtract(m1, m2)
```



Note that if you want to apply one of the feedback operators in a situation where there is only one machine, you can use `sm.Gain(1.0)` or `sm.Wire()` as the other argument.

Check Yourself 3. Use gains, delays, and adders to draw a system diagram for the first system in tutor problem Wk.4.2.1. (That is, the tutor problem that you did before coming to lab).

See sheet

Check Yourself 4. Use gains, delays, and adders to draw a system diagram for the second system in tutor problem Wk.4.2.1.

Check Yourself 5. Use gains, delays, and adders to draw a system diagram for the third system in tutor problem Wk.4.2.1.

Wk.4.3.2 Do tutor problem Wk.4.3.2.

Check Yourself 6. Use gains, delays, and adders to draw a system diagram for the **controller** in the wall-finder system.

Check Yourself 7. Use gains, delays, and adders to draw a system diagram for the **plant** in the wall-finder system.

Check Yourself 8. Use gains, delays, and adders to draw a system diagram for the **sensor** in the wall-finder system.

Check Yourself 9. Connect the previous three component systems to make a diagram of the wall-finder system. Label all the wires. Draw boxes around the controller, plant, and sensor components.

Checkoff 1. Show your system diagrams to a staff member. Identify the instances of cascade and feedback composition for the wall-finder system.

Wk.4.3.4 Do tutor problem Wk.4.3.4. We encourage you to write your code in `dl4Work.py` and test it within Idle before copying it into the Tutor

With a value of `T = 0.1` and an initial distance to the wall of 1.5 meters, experiment with different values of the gain. You can do this using the `plotD` procedure, defined in `dl4Work.py`. For a given gain value, `k`, it will make a plot of the sequence of distances to the wall. Be sure to use your code running within Idle.

Checkoff 2. Find three different values of `k`, one for which the distance converges monotonically, one for which it oscillates and converges, and one for which it oscillates and diverges. Show plots for each of these `k` values to a staff member. Save screen shots for each of these plots (you can find instructions under the Reference tab of our home page.)

Wk.4.3.6 Enter the gains you found into the tutor.

4 On the simulated robot

Implement a brain for the wall-finder problem using a state machine, as described in **Section 2.**

Recall that the robot itself is the plant and so we do not need to write any code for that. We have already implemented a `Sensor` state machine which outputs a delayed version of the values of sonar sensor 3 (the robot actually has relatively little delay in its sensing).

Your job is to implement the `Controller` state machine, which takes as input the output of the sensor state machine, and generates as output instances of `io.Action` with 0 rotational velocity and an appropriate forward velocity. It should depend on `dDesired`, which is the desired distance to the wall. Do this by editing the `getNextValues` method of the `Controller` class in `Desktop/6.01/lab4/designLab/smBrainPlotDistSkeleton.py`. Your controller should attempt to make the output of sonar sensor 3 be equal to 0.7, even though sensor 3 doesn't point straight forward.

For each of the three gains you found in Checkoff 2, run the simulated robot in the simulated world specified in `Desktop/6.01/lab4/designLab/worlds/wallFinderWorld.py`. Whenever you start or reset the world, the robot will be 1.5 meters from the wall it is facing. Save the plot from each of these runs. You can find instructions for saving plots as screen shots under the Reference tab of our home page.

Checkoff 3. Show your plots of the robot simulations from each of runs to a staff member. Compare them to the plots you got by running the state machine models. Explain how they differ and speculate about why. Email your code and plots to your partner.

Design Lab 4

9/30

4.3.1

Controller

$V = kE = \cancel{D_1[n] - D_0[n-1]}$

C = none ✓

$d = \frac{1}{5}E$ ✗

.2 ✗

0 ✗

5 ✗ = ~~###~~ duh $5 = k \cdot 1$

negitive since getting closer to wall

-5 ✓

Plant

none ✗

$-.5$ ✗

$= -5 \cdot .1$ ✗

$-.5$ ✗

input is just command vel + rot

-5

none ✗

output's new robot actual position

D_0 plays a role

$D_0[n-1]$

$d[n] = \cancel{5} \cdot V[n]$ controller w/ factor by k

$V[n]$

(2)

↓ Controller sends, all scaled

$d_0[n] = T \cdot V[n] + D_0[n-1]$

$T = .1$

$V = X = kE$

k is -5

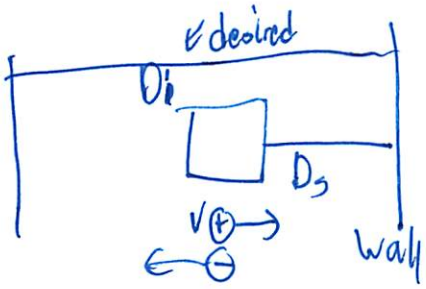
$V = -5E$

$-5Et$ | .1 ✓

$+$

$V[n] = X$

.1 ✗



$D_0[n] = \overbrace{D_0[n-1]}^{=D_s} \pm T \cdot V[n-1]$

want to inc. (distance from wall)

want −

~~+~~

← D_f .1 ✗

(3)

(-)

?

want to

subtract a negitive

0 -.1 ✓

Sensor

$$D_s[n] = D_o[n-1]$$

none ✗ 1 ✗ just a delay

1 ✗ none ✓

based on problem #; not based on previous output

.9 ✗

0 1 ✓

Combined difference eq

$$D_o[n] = D_o[n-1] - T \cdot V[n-1]$$

actual pos

need a D!?

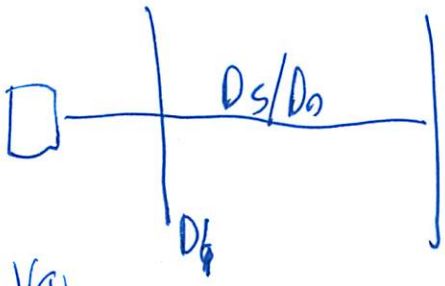
(4)

$V = kE$

$= k[D_i - D_o[n-1]]$

$D_o[n] = D_o[n-1] - T \cdot k \cdot (D_i[n] - D_o[n-1])$

VHD - wrong



V⊕

D_o = fixed — ⊕

D_o dec↓

$-1(.5 \cdot 1)$

$-.5$

$1 - .1 \cdot 5 \cdot 1)$

$.5$

$D_o[n] = D_o[n-1] - T \cdot k \cdot (D_i[n-1] - D_o[n-2])$

$0 \quad -.5$ ⊗

$k = -5$

$0 \quad .5$ ✓

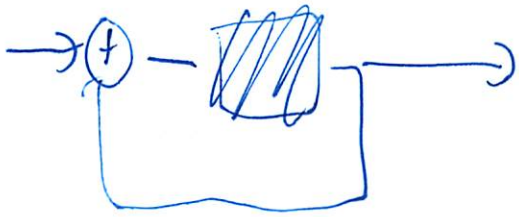
$1 \quad +.5$ ⊗

$-1 \cdot -.5 \cdot -1$

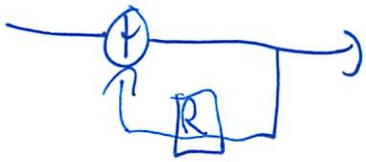
$1 \quad -.5$ ✓

(5)

CY #3

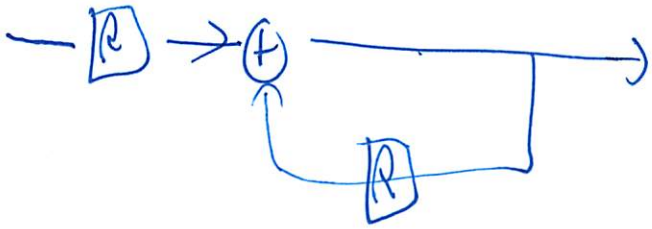


(X) infinte loop



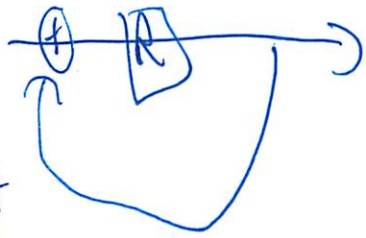
(✓)

CY #4

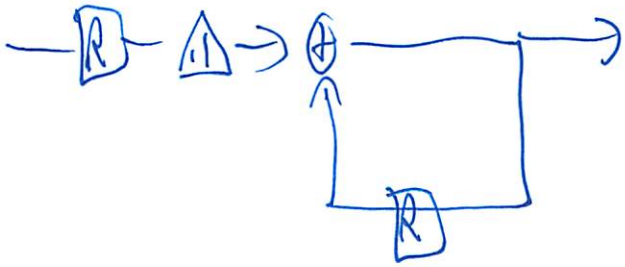


(J) =

equivilant (trick)

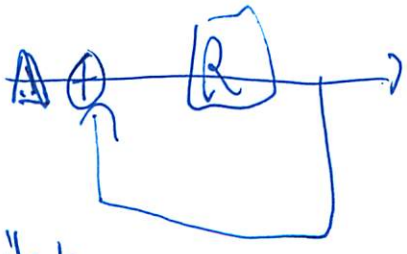


CY #5



(J) =

equivilant (trick)



(6) 4.3.2 Tutor

State machine composition

proc accumulator( )
↑ output before 1st input

- like problem
  sum n upto + incl'ing n
- combo of SMs
- kinda easy
- got 1st feedback add (wire, R(init))

2nd one ✓
feedback add (R(init), wire(1)) our little trick

3rd
feedback add
I would use a cascade
~~Cascade~~(Cascade (R(init), Gain(.1)), ~~Feedback~~
Cascade(↓ , feedback add (↘))

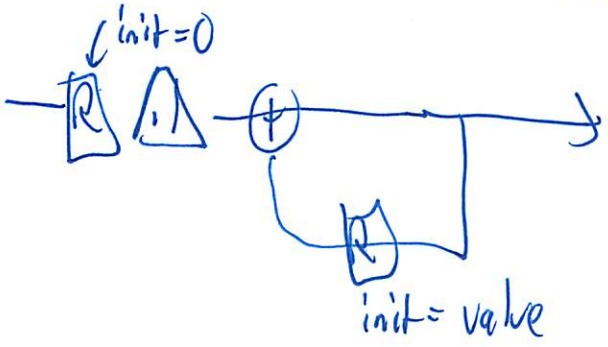
- close but wrong
- first is correct
  - I think 2nd init needs to be 0 ↗
    - nope

(7)

- also want s not hardcoded. I
  - but they were testing w/ .1
- for 2nd Q('init • s)
  - no again

I am overshooting by .5

- want Q on bottom, 2nd argument
- not out little trick
- now everything is 1 too big!
  - So 1st delay init should be 0



Why? - way problem is defined I guess first value is 0 - pass on that

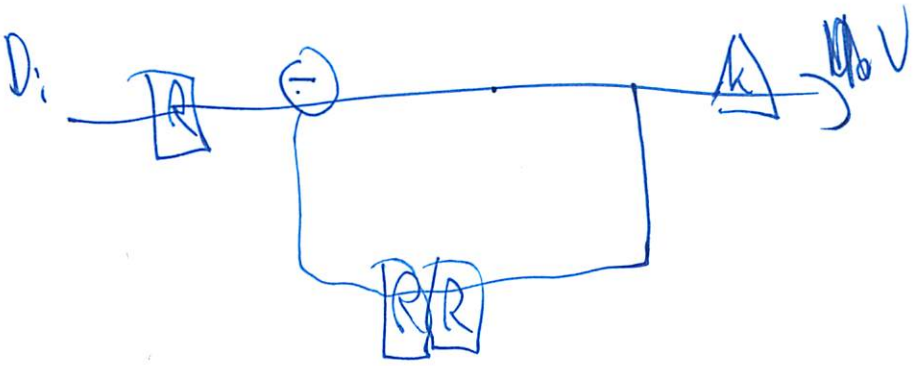
prof: make table to check

CY6 Controller

(8)

$D_o =$

$V = k\left[D_i[n-1] - D_o(n-2)\right]$ ← other port



input = E

$V = k[$

D_i

$V = kE$

$E[n-1]$

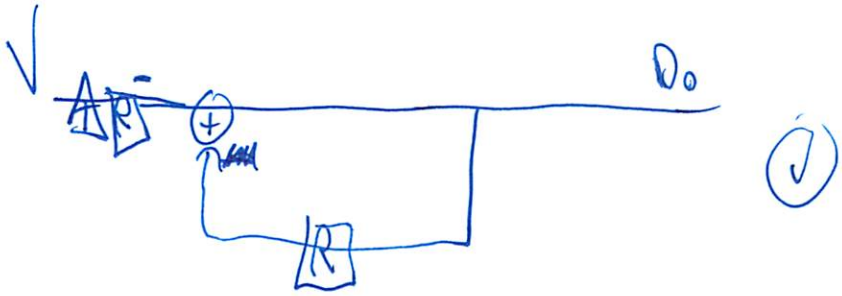
$k(D_i = kE(n))$

(V)

(9) CY7 Plant

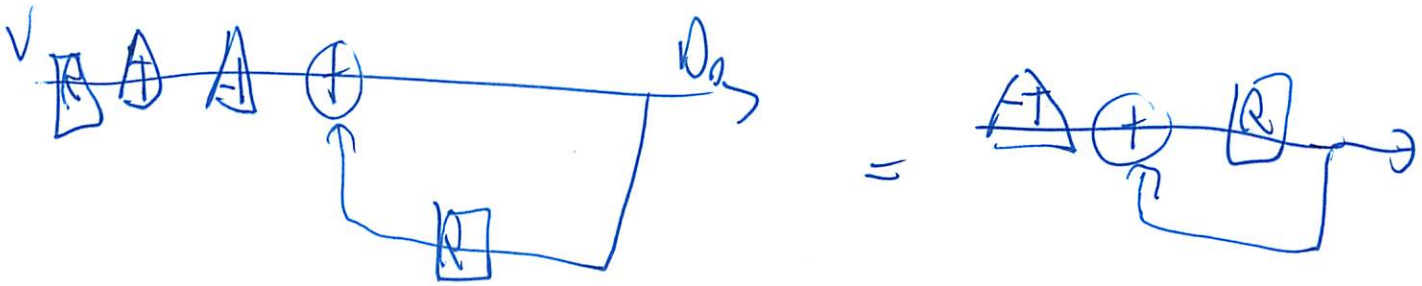
$$D_o[n] = D_o[n-1] \pm \tau \cdot V[n-1]$$

($D_o[n-1]$: output; $V[n-1]$: input)



(J)

P_r



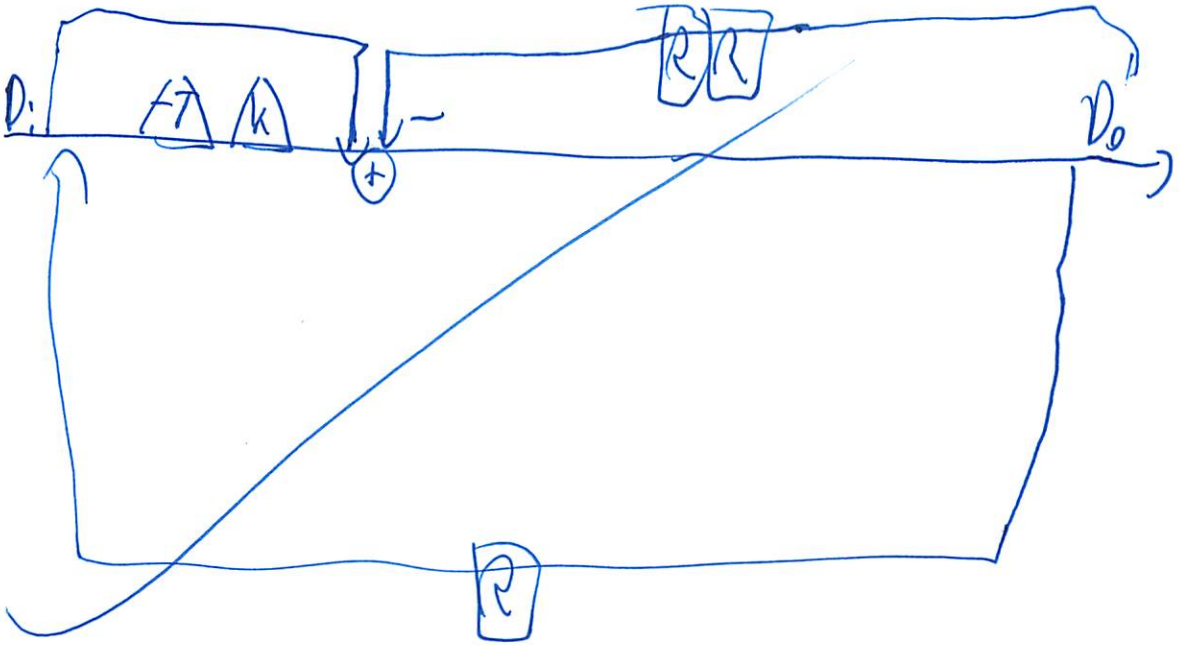
CY8 Sensor

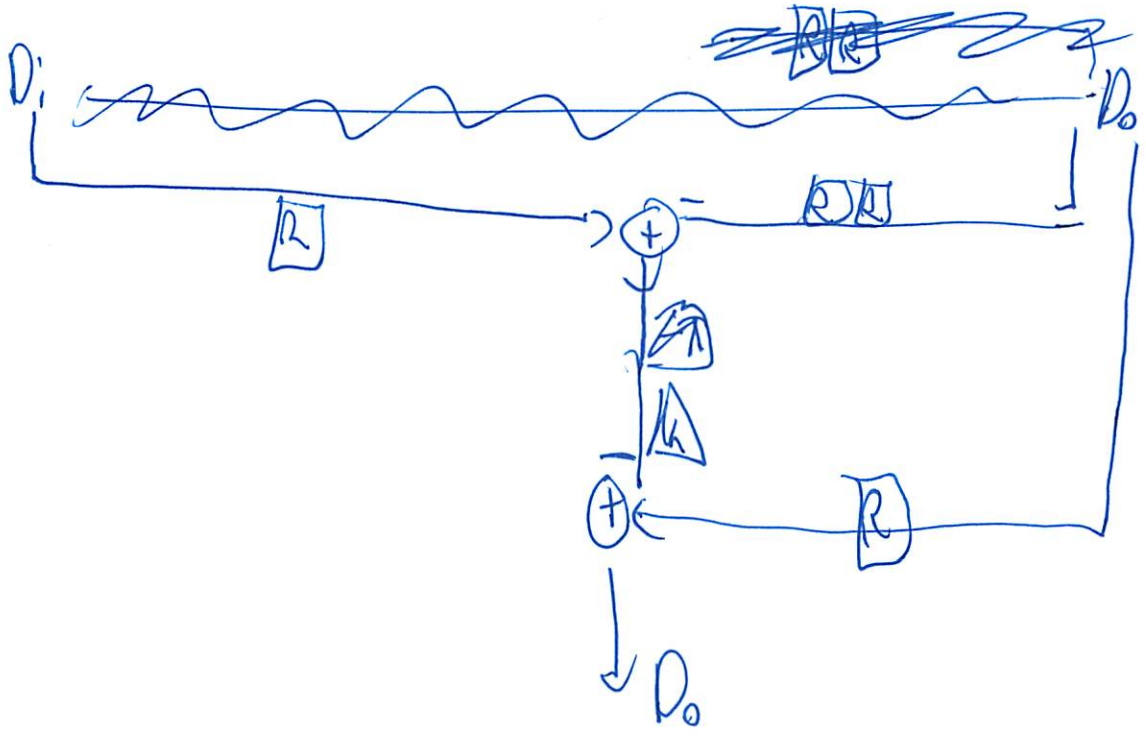
$$D_s[n] = D_o[n-1]$$

($D_o[n-1]$: input)

(10) C4P Whole Thing

Thought through ea and try to draw



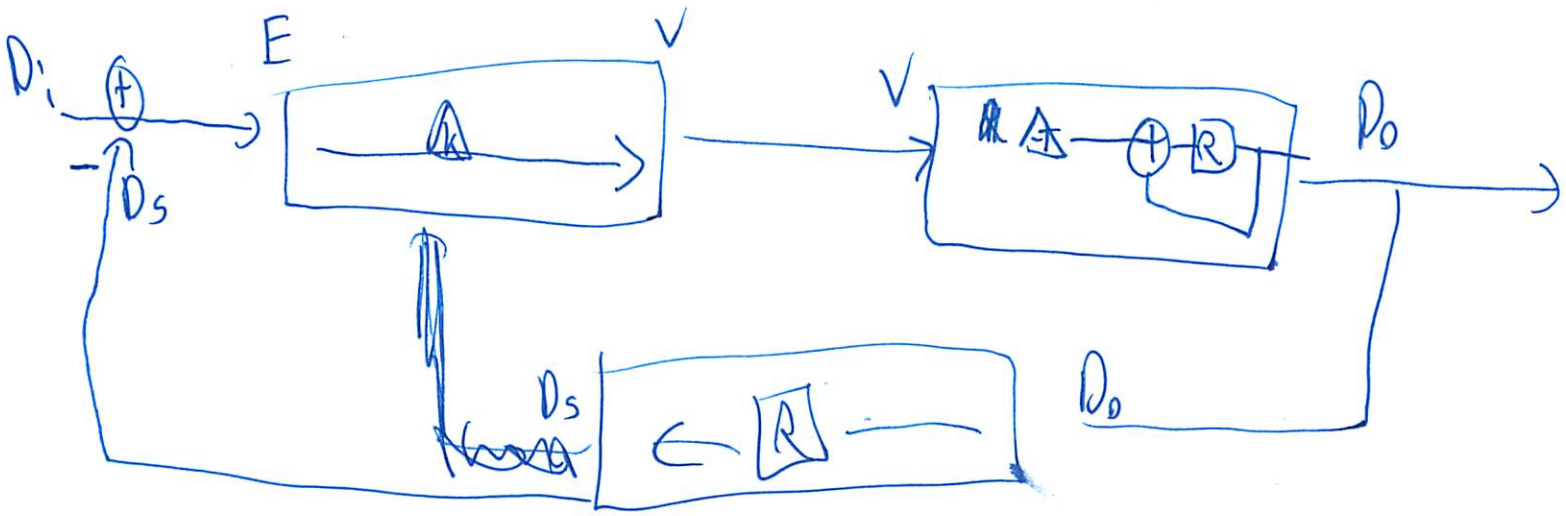


think boxes togeter

fill in each of boxes w/ sub steps

↓

(11)



4.3.4 Now back to wall finder → code it up

Controller procedure

- oh just implement diagram

Sm. Gain (k)

plant

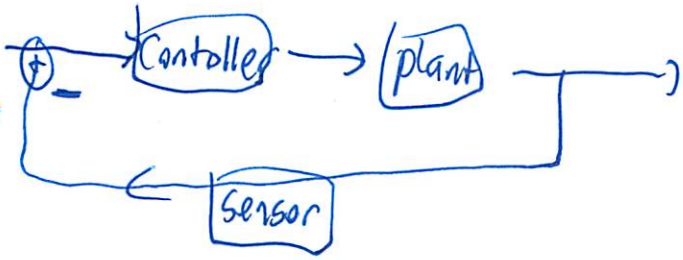
Cascade (Gain(-1), Feedback add (R(initD), Wire()))

Sensor

Sm. ~~Gain~~ R(initD)

System

use componant parts



Feedback(Subtract(Cascade(Controller, Plant), Sensor)

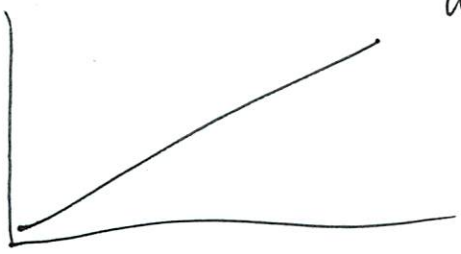
(12) Tutor check ✓bingo
-all for today

Partner dropped 10/4

T = .01
inital distance = 1.5
experiment w/ plot
plot w/ different k (gain values)
-give the k

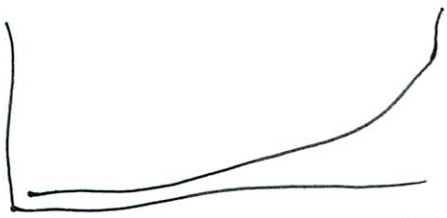
1

almost linearly

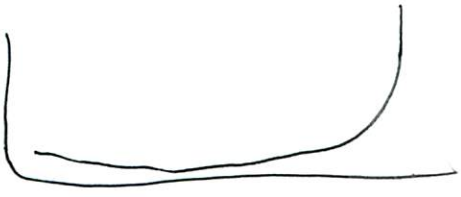


1.2

1

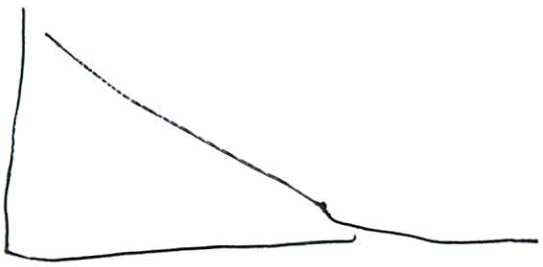


1.5



(13)

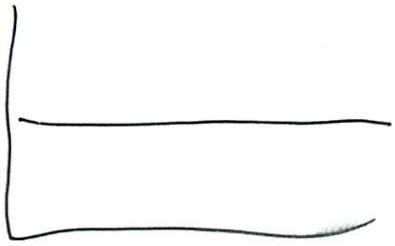
-.1



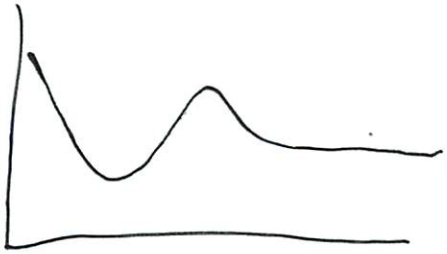
converging monotonically

-1 is better

0



-.5



k = gain

= how it responds to error

oscilate + converge

-15

oscilate + diverge

Check off 2 ✓

factor -1
-5
-15

(19

Last step!

- implement Controller sm get next value

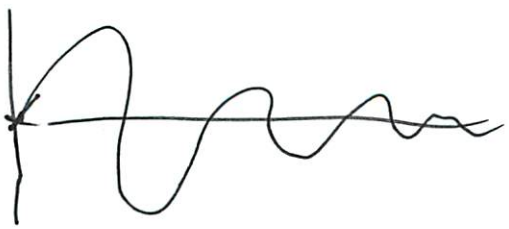
sensor[3] should = .7

- not in a thinking mood
  - can I get done in 40 min?

- copy in robot code
  - no we just want controller
  - that was representation
- inp is already sensor[3]
- build non isolation in?
  - where change gain?
- want it to calc error
  - a gain
  - that is ✓
- oh I see
  - now it stops at pt
    - ↳ converges

- So the sensor reading is like the graph

k = -5



(15) on -15

-Crases + max sensor reading
= 5

so in accurate

Hw due wed night
-total wrong position

6.01: Introduction to EECS I

Predicting System Performance

Week 5 *October 5, 2010*

Outline

- System functions: primitives and compositions
- Modes of feedback systems
- Finding and interpreting poles

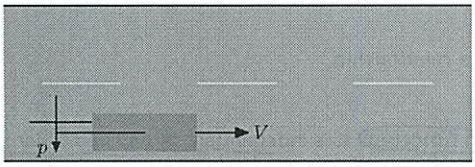
Reading: Chapter 6

Midterm exam:

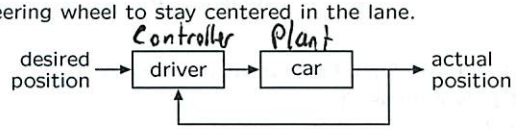
- Tuesday, October 12, 7:30–9:00 PM
- 32-141 or 32-155
- Any printed material okay
- No computers or phones
- Make-up: Wednesday, October 13, 8:00–9:30 AM; email `welg@mit.edu`

Feedback and Control

Feedback is pervasive in natural and artificial systems.

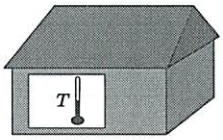


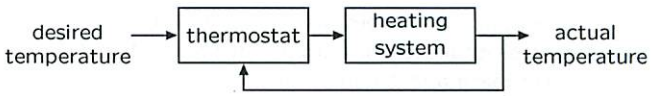
Turn steering wheel to stay centered in the lane.



Feedback and Control

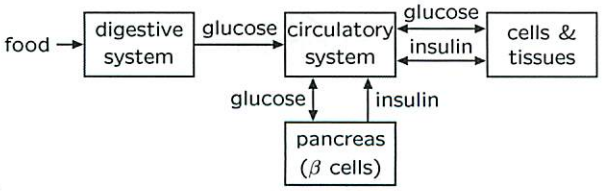
Feedback is useful for regulating a system's behavior, as when a thermostat regulates the temperature of a house.

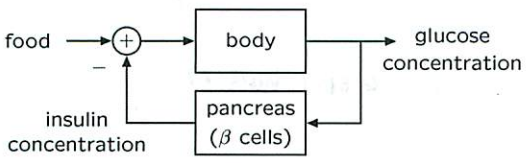




Feedback and Control

Concentration of glucose in blood is highly regulated and remains nearly constant despite episodic ingestion and use.

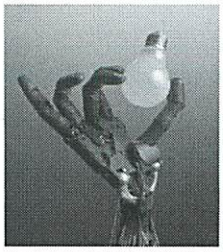




Feedback and Control

Motor control relies on feedback from pressure sensors in the skin as well as proprioceptors in muscles, tendons, and joints.

Try building a robotic hand to unscrew a lightbulb!



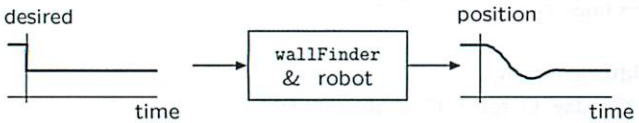
Shadow Dexterous Robot Hand (Wikipedia)

Performance analysis

quantify

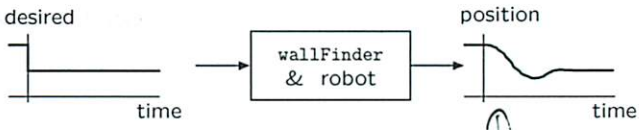
We can quantify the performance of a system by characterizing the **signals** that the system generates.

properties of system



- could difference eq, input + output

Important features of behavior



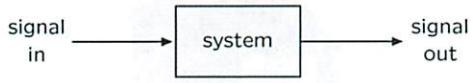
What are the important **features** of this output signal?

overshoot
delay
settling time

Overshoot and then go back
- ~~delay~~ to pt time
- Convergence

Analyzing systems

Our goal is to develop representations for systems that facilitate analysis.



Examples:

- Does the output signal overshoot? If so, how much?
- How long does it take for the output signal to reach its final value?
- Does it ever reach a stable final value?

System functions

Any LTI system is completely characterized by the relationship between the input signal X and the output signal Y.

We call this relationship,

$$H = \frac{Y}{X}$$

the **system function**. It is independent of any particular input signal, just as a mathematical function or a Python procedure is an entity, independent of its arguments. - since linear

System functions for LTI systems are always ratios of polynomials in $\mathcal{R}$.

$\mathcal{R}$ = delay, adders, etc

(output only depends on previous inputs, not on any outputs

Feed-forward SFs

Gain: $Y = kX$

$$H = \frac{Y}{X} = k$$

Delay: $Y = \mathcal{R}X$

$$H = \frac{Y}{X} = \mathcal{R}$$

Combinations: $Y = 3\mathcal{R}^2X - \mathcal{R}X + 2X$

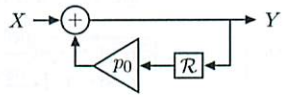
$$H = \frac{Y}{X} = 3\mathcal{R}^2 - \mathcal{R} + 2$$

System functions for feed-forward systems have 1 in the denominator.

Feedback SFs

Consider a simple system with **feedback**.

output matters here



$$Y = X + p_0\mathcal{R}Y$$

$$(1 - p_0\mathcal{R})Y = X$$

$$Y = \frac{X}{1 - p_0\mathcal{R}}$$ ← solve for y

System function

$$H = \frac{Y}{X} = \frac{1}{1 - p_0\mathcal{R}}$$ ← still have on

- oh that is difference
- never thought of it that way

Feedback

The reciprocal of $1 - p_0\mathcal{R}$ can be evaluated using synthetic division.

$$
\begin{array}{r|l}
1 - p_0\mathcal{R} & 1 + p_0\mathcal{R} + p_0^2\mathcal{R}^2 + p_0^3\mathcal{R}^3 + \cdots \\
\hline
1 & \\
1 - p_0\mathcal{R} & \\
\hline
p_0\mathcal{R} & \\
p_0\mathcal{R} - p_0^2\mathcal{R}^2 & \\
\hline
p_0^2\mathcal{R}^2 & \\
p_0^2\mathcal{R}^2 - p_0^3\mathcal{R}^3 & \\
\hline
p_0^3\mathcal{R}^3 & \\
p_0^3\mathcal{R}^3 - p_0^4\mathcal{R}^4 & \\
\hline
\cdots &
\end{array}
$$

Therefore

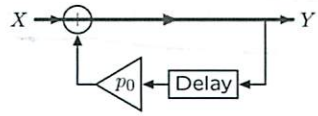
$$\frac{1}{1 - p_0\mathcal{R}} = 1 + p_0\mathcal{R} + p_0^2\mathcal{R}^2 + p_0^3\mathcal{R}^3 + p_0^4\mathcal{R}^4 + \cdots$$

(show do you know how to evaluate?

Feedback: Cyclic signal flow paths

Feedback implies cyclic signal flow paths.

$x[n] = \delta[n]$

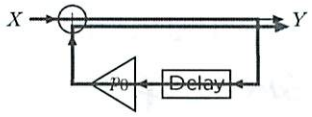


$$\frac{Y}{X} = \frac{1}{1 - p_0\mathcal{R}} = 1 + \cdots$$

Feedback: Cyclic signal flow paths

Feedback implies cyclic signal flow paths.

$x[n] = \delta[n]$

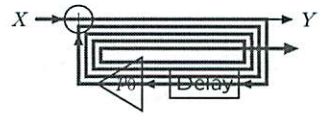


$$\frac{Y}{X} = \frac{1}{1 - p_0\mathcal{R}} = 1 + p_0\mathcal{R} + \cdots$$

Feedback: Cyclic signal flow paths

Feedback implies cyclic signal flow paths.

$x[n] = \delta[n]$



$$\frac{Y}{X} = \frac{1}{1 - p_0\mathcal{R}} = 1 + p_0\mathcal{R} + p_0^2\mathcal{R}^2 + p_0^3\mathcal{R}^3 + p_0^4\mathcal{R}^4 + \cdots$$

i^2 since multiplied each time it goes around

All cyclic paths must contain at least one delay.

modeling from start, empty time

Example: Accumulator

$$\frac{Y}{X} = \frac{1}{1 - \mathcal{R}} = 1 + \mathcal{R} + \mathcal{R}^2 + \mathcal{R}^3 + \mathcal{R}^4 + \cdots$$

If input is non-zero starting at time zero, then can see output will be governed by

$1, 1 + \mathcal{R}, 1 + \mathcal{R} + \mathcal{R}^2, 1 + \mathcal{R} + \mathcal{R}^2 + \mathcal{R}^3, \cdots$

Thus, if input is

$x[n] = \delta[n]$

$1, 1, 1, 1, \cdots$

If input is

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

$1, 2, 3, 4, \cdots$

diff ways to think about it

System functions for LTI systems

Ratio of polynomials in $\mathcal{R}$:

$$y[n] + a_1 y[n-1] + a_2 y[n-2] + a_3 y[n-3] + \cdots = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + b_3 x[n-3] + \cdots$$

general form

$$(1 + a_1\mathcal{R} + a_2\mathcal{R}^2 + a_3\mathcal{R}^3 + \cdots)\, Y = (b_0 + b_1\mathcal{R} + b_2\mathcal{R}^2 + b_3\mathcal{R}^3 + \cdots)\, X$$

$$D(\mathcal{R})\, Y = N(\mathcal{R})\, X$$

$$\frac{Y}{X} = \frac{N(\mathcal{R})}{D(\mathcal{R})} = \frac{b_0 + b_1\mathcal{R} + b_2\mathcal{R}^2 + b_3\mathcal{R}^3 + \cdots}{1 + a_1\mathcal{R} + a_2\mathcal{R}^2 + a_3\mathcal{R}^3 + \cdots}$$ feed forward part

Persistent part of response of such a system is associated with denominator.

Example $\frac{Y}{X} = \frac{1}{1+2\mathcal{R}} = 1 - 2\mathcal{R} + 4\mathcal{R}^2 - 8\mathcal{R}^3$ or convert to diff eq

walk through division $Y = -2\mathcal{R}Y + X$

$y[n] = -2y[n-1] + x[n]$

System functions: Why do we care

PCAP system on system functions makes it easier to combine models than manipulating systems of operator equations.

System functions expose important analytic properties of the system.

PCAP: Primitive SFs

Gain: $Y = kX$

$$H = \frac{Y}{X} = k$$

Delay: $Y = \mathcal{R}X$

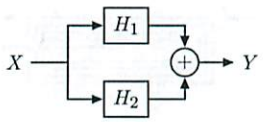
$$H = \frac{Y}{X} = \mathcal{R}$$

closure: adding system function outputs polynomials in $\mathcal{R}$

Combining SFs: Sum

The system function of the **sum of two systems** is the sum of their system functions. This relies on properties of ratios of polynomials: their sum can be written as a ratio of polynomials.

$Y_1 = H_1X \qquad Y_2 = H_2X$



$$\begin{aligned} Y &= Y_1 + Y_2 \\ &= H_1X + H_2X \\ &= (H_1 + H_2)X \\ &= HX \end{aligned}$$

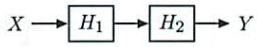
where $H = H_1 + H_2$.

- already done abstraction
- don't know what's in the box

Combining SFs: Cascade

The system function of the **cascade of two systems** is the product of their system functions. This relies on properties of ratios of polynomials: their product can be written as a ratio of polynomials.

$W = H_1X \qquad Y = H_2W$



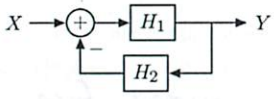
$$\begin{aligned} Y &= H_2W \\ &= H_2H_1X \\ &= HX \end{aligned}$$

where $H = H_2H_1$.

Same as if reversed order for LTI

Combining SFs: Negative feedback

Concentrate on **negative feedback** and **Black's formula**:



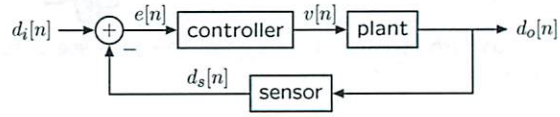
$$\begin{aligned} Y &= H_1(X - H_2Y) \\ Y + H_1H_2Y &= H_1X \\ Y(1 + H_1H_2) &= H_1X \\ Y &= \frac{H_1}{1 + H_1H_2}X \\ Y &= HX \end{aligned}$$

where

$$H = \frac{H_1}{1 + H_1H_2}.$$

Wall finder

Control the robot to move to desired distance from a wall.



Controller (brain): sets velocity $\propto$ error

$V = KE$ proportional

Plant (robot locomotion): integrates velocity to get position:

$D_o = \mathcal{R}D_o - TRV$ w/ accumulator

Sensor: introduces a delay

$D_s = \mathcal{R}D_o$

$1 - \frac{d_{n-1}}{d_n}$

current pos is $d_n - d_{n-1} = -TRV$

how far it moved that step / old vel - flag

Use composition to find SF

Controller

$$H_c = \frac{V}{E} = K$$

Plant

$$H_p = \frac{D_o}{V} = \frac{-T\mathcal{R}}{1-\mathcal{R}}$$

Sensor

$$H_s = \frac{D_s}{D_o} = \mathcal{R}$$

Cascade: Controller and Plant

$$H_{cp} = \frac{D_o}{E} = \frac{-KT\mathcal{R}}{1-\mathcal{R}}$$

Negative feedback: ControllerPlant and Sensor

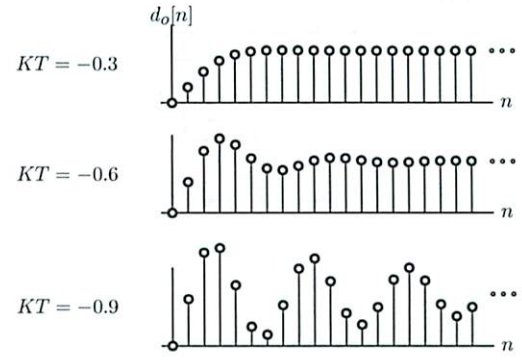
$$H = \frac{D_o}{D_i} = \frac{\frac{-KT\mathcal{R}}{1-\mathcal{R}}}{1+\mathcal{R}(\frac{-KT\mathcal{R}}{1-\mathcal{R}})} = \frac{-KT\mathcal{R}}{1-\mathcal{R}-KT\mathcal{R}^2}$$

characterizes what robot will do

- convert to diff eq + try samples

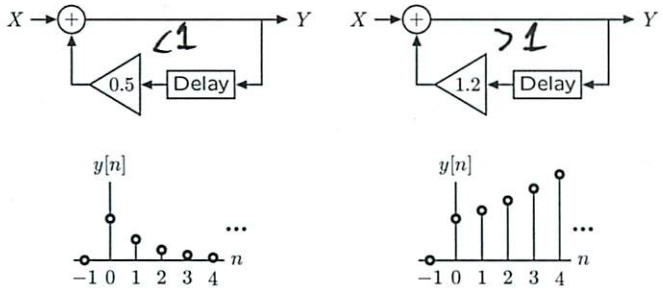
Wall finder

The behavior of the system depends critically on KT.



Geometric growth

If traversing the cycle decreases or increases the magnitude of the signal, then the output will decay or grow, respectively.



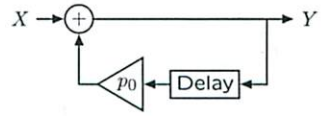
Unit sample response:

→ geometric sequences: $y[n] = (0.5)^n$ and $(1.2)^n$ for $n \geq 0$.

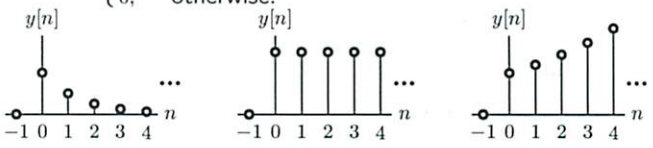
Geometric growth

These system responses can be characterized by a single number (the pole), which is the base of the geometric sequence.

↑ long term behavior



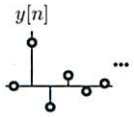
$$y[n] = \begin{cases} p_0^n, & \text{if } n >= 0; \\ 0, & \text{otherwise.} \end{cases}$$



$p_0 = 0.5$ $p_0 = 1$ $p_0 = 1.2$

Check Yourself

What value of p_0 represents the signal below?



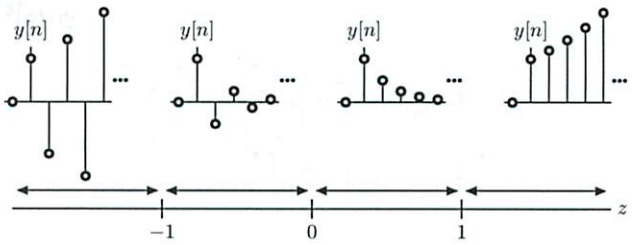
1. $p_0 = 0.5$
2. $p_0 = -0.5$ ← only will alternate if negitive since
3. $p_0 = 0.25$ interspersed with $p_0 = -0.25$

if. ... − + − +

$n \Theta \rightarrow n^1\ n^2\ n^3\ n^4$

Geometric growth

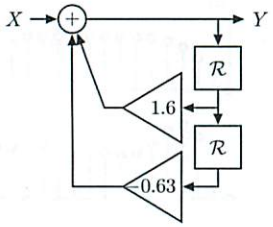
The value of p_0 determines the rate of growth.



$p_0 < -1$: magnitude diverges, alternating sign
$-1 < p_0 < 0$: magnitude converges, alternating sign
$0 < p_0 < 1$: magnitude converges monotonically
$p_0 > 1$: magnitude diverges monotonically

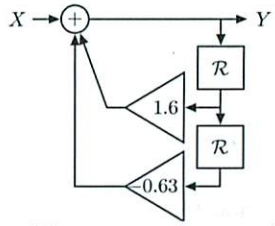
Second-order systems

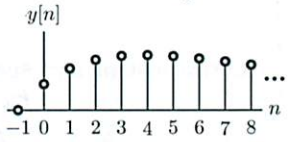
The unit-sample response of more complicated cyclic systems is more complicated.



Second-order systems

The unit-sample response of more complicated cyclic systems is more complicated.

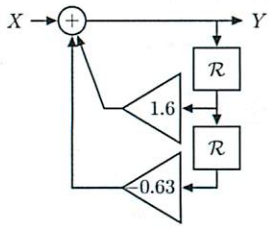


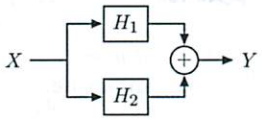


Not geometric. This response grows then decays.

Second-order systems: Additive decomposition

This system function can be written as a sum of simpler parts.





$$Y = X + 1.6\mathcal{R}Y - 0.63\mathcal{R}^2Y$$

$$(1 - 1.6\mathcal{R} + 0.63\mathcal{R}^2)\,Y = X$$

$$\frac{Y}{X} = \frac{1}{1-1.6\mathcal{R}+0.63\mathcal{R}^2} = \frac{1}{(1-0.9\mathcal{R})(1-0.7\mathcal{R})}$$

factor + solve

Additive decomposition: partial fraction expansion

$$\frac{1}{(1-0.9\mathcal{R})(1-0.7\mathcal{R})} = \frac{A}{1-0.9\mathcal{R}} + \frac{B}{1-0.7\mathcal{R}}$$

Cross multiply

$$1 = A(1-0.7\mathcal{R}) + B(1-0.9\mathcal{R})$$

Collect like terms

$$1 = (A+B) - (0.7A + 0.9B)\mathcal{R}$$

So, we have

$$1 = A + B$$

$$0 = 0.7A + 0.9B$$

So, $A = 4.5$ and $B = -3.5$, and

$$\frac{Y}{X} = \frac{4.5}{1-0.9\mathcal{R}} - \frac{3.5}{1-0.7\mathcal{R}}$$

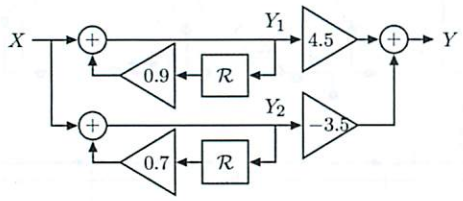
feedback

lots of math!

Second-order systems: Additive decomposition

$$\frac{Y}{X} = \frac{4.5}{1-0.9\mathcal{R}} - \frac{3.5}{1-0.7\mathcal{R}}$$

same system



Second-order systems: Additive decomposition

Operation of sum of systems on a signal S

$$(H_1 + H_2)S = H_1S + H_2S$$

Our system

$$H = \frac{Y}{X} = \frac{4.5}{1-0.9\mathcal{R}} - \frac{3.5}{1-0.7\mathcal{R}} = H_1 + H_2$$

Unit sample response:

- to H_1 is a geometric sequence.
- to H_2 is a geometric sequence.
- to $H = H_1 + H_2$ is the sum of geometric sequences.

Actual values

If $x[n] = \delta[n]$ then $y_1[n] = 0.9^n$ and $y_2[n] = 0.7^n$ for $n \geq 0$.

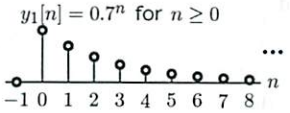
Thus, $y[n] = 4.5(0.9)^n - 3.5(0.7)^n$ for $n \geq 0$.

Mode with biggest base eventually governs behavior

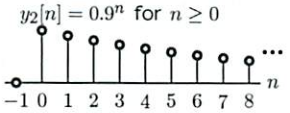
Sum of geometric sequences

Mode with biggest base eventually governs behavior

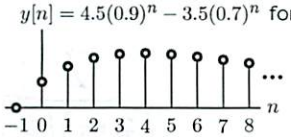
$y_1[n] = 0.7^n$ for $n \geq 0$



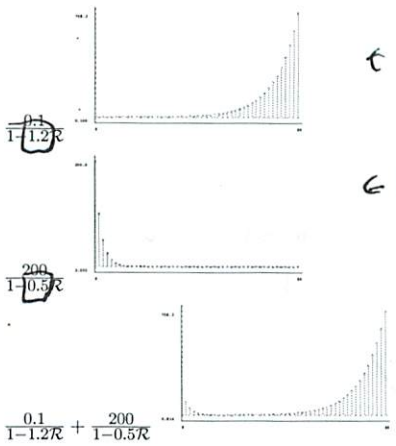
$y_2[n] = 0.9^n$ for $n \geq 0$



$y[n] = 4.5(0.9)^n - 3.5(0.7)^n$ for $n \geq 0$



More dramatically

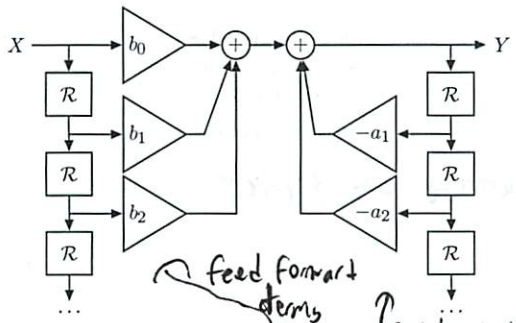


eventually big base wins

large coefficient dominates 1st

Analysis of more complicated systems

Rational polynomials can be realized with block diagrams of the following form:



feed forward terms

feedback loops

$$\frac{Y}{X} = \frac{b_0 + b_1\mathcal{R} + b_2\mathcal{R}^2 + b_3\mathcal{R}^3 + \cdots}{1 + a_1\mathcal{R} + a_2\mathcal{R}^2 + a_3\mathcal{R}^3 + \cdots}$$

Analysis of more complicated systems

Modes can be identified by expanding system functional in partial fractions.

$$\frac{Y}{X} = \frac{b_0 + b_1\mathcal{R} + b_2\mathcal{R}^2 + b_3\mathcal{R}^3 + \cdots}{1 + a_1\mathcal{R} + a_2\mathcal{R}^2 + a_3\mathcal{R}^3 + \cdots}$$

Factor denominator:

$$\frac{Y}{X} = \frac{b_0 + b_1\mathcal{R} + b_2\mathcal{R}^2 + b_3\mathcal{R}^3 + \cdots}{(1 - p_0\mathcal{R})(1 - p_1\mathcal{R})(1 - p_2\mathcal{R})(1 - p_3\mathcal{R})\cdots}$$

can always factor

Partial fractions:

$$\frac{Y}{X} = \frac{C_0}{1 - p_0\mathcal{R}} + \frac{C_1}{1 - p_1\mathcal{R}} + \frac{C_2}{1 - p_2\mathcal{R}} + \cdots + D_0 + D_1\mathcal{R} + D_2\mathcal{R}^2 + \cdots$$

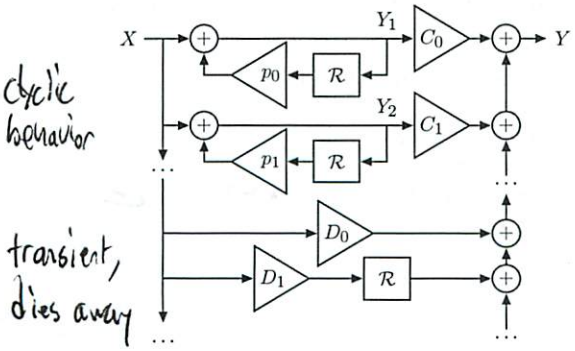
One mode of the form p_i^n arises from each factor of the denominator.

might need matlab to solve

Analysis of more complicated systems

Modal decomposition provides an alternative block diagram.

can rewrite in this form



cyclic behavior

transient, dies away

The upper part is cyclic; the lower part is acyclic.

Easy way to find poles

The poles are also the roots of the denominator of the system functional when $\mathcal{R} \to \frac{1}{z}$.

Start with system functional:

$$\frac{Y}{X} = \frac{1}{1 - 1.6\mathcal{R} + 0.63\mathcal{R}^2} = \frac{1}{(1 - p_0\mathcal{R})(1 - p_1\mathcal{R})} = \frac{1}{\underbrace{(1 - 0.7\mathcal{R})}_{p_0=0.7}\underbrace{(1 - 0.9\mathcal{R})}_{p_1=0.9}}$$

Substitute $\mathcal{R} \to \frac{1}{z}$ and find roots of denominator:

$$\frac{Y}{X} = \frac{1}{1 - \frac{1.6}{z} + \frac{0.63}{z^2}} = \frac{z^2}{z^2 - 1.6z + 0.63} = \frac{z^2}{\underbrace{(z - 0.7)}_{z_0=0.7}\underbrace{(z - 0.9)}_{z_1=0.9}}$$

multiply by z^2

poles are at 0.7 and 0.9

quadratic eq

Check Yourself

Consider the system described by

$$y[n] = -\frac{1}{4}y[n-1] + \frac{1}{8}y[n-2] + x[n-1] - \frac{1}{2}x[n-2]$$

How many of the following are true?

1. The unit sample response converges to zero. ✓
2. There are poles at $z = \frac{1}{2}$ and $z = \frac{1}{4}$.
3. There is a pole at $z = \frac{1}{2}$.
4. There are two poles. ✓

Properties of signals

A signal is **transient** if it has finitely many non-zero samples. Otherwise, it is **persistent**.

input/output

A signal is **bounded** if there is exist upper and lower bound values such that the samples of the signal never exceed those bounds; otherwise it is **unbounded**.

- A transient input to an acyclic (feed-forward) system results in a transient output.
- A transient input to a cyclic (feed-back) system results in a persistent output.

$Y = \frac{1}{4}RY + \frac{1}{8}R^2Y + RX - \frac{1}{2}R^2X$

$Y(1 + \frac{1}{4}R - \frac{1}{8}R^2) = X(R - \frac{1}{2}R^2)$

$\frac{Y}{X} = \frac{R(1-\frac{1}{2}R)}{1+\frac{1}{4}R-\frac{1}{8}R^2} \Rightarrow \frac{\frac{1}{z}(1-\frac{1}{2}\frac{1}{z})}{1+\frac{1}{4}\frac{1}{z}-\frac{1}{8}\frac{1}{z^2}} = \frac{z-\frac{1}{2}}{z^2+\frac{1}{4}z-\frac{1}{8}}$

$\frac{z-\frac{1}{2}}{(z+\frac{1}{2})(z-\frac{1}{4})}$

Poles: Summary

- The **poles** of a system are the roots of the denominator polynomial of the system function in $1/\mathcal{R}$.
- The **dominant pole** is the pole with the largest magnitude.

sign does not matter

Dependence on pole magnitude

Response to a bounded input signal, if the dominant pole has magnitude

- > 1: output signal will be unbounded
- < 1: output signal will be bounded
  if the input is transient, output signal will converge to 0.
- 1: output signal will be bounded

A system is *stable* if the output signal is bounded.

Characterize any system

Dependence on pole type

Response to a transient input signal, if the dominant pole is

- **real and positive**: output signal will, after finitely many steps, begin to increase or decrease monotonically. no oscilate
- **real and negative**: output signal will, after finitely many steps, begin to alternate signs. oscilate
- **complex**: output signal will, after finitely many steps, begin to be periodic, with a period of $2\pi/\Omega$, where Ω is the 'angle' of the pole in the complex plane. Oscilate

<1 periodic → converge to 0

>1 periodic → unbounded

This Week

Software lab: Class for manipulating system functions

Design lab: Analyzing robot control system for stability

Midterm exam:

- Tuesday, October 12, 7:30–9:00 PM
- 32-141 or 32-155
- Any printed material okay
- No computers or phones
- Make-up: Wednesday, October 13, 8:00–9:30 AM; email `welg@mit.edu`

Software Lab 5: System Functions

1 Setup

- **Using a lab laptop or desktop machine**: Log in using your Athena user name and password; in the terminal window, type `athrun 6.01 update`. Work in the directory `Desktop/6.01/lab5/swLab`.
- **Using your own laptop**: Download the zip file for software lab 5 from the *Calendar* tab of the course web page, unzip it, and work in the resulting directory.

2 System Function Class

The software lab for this week is to implement a Python class that represents LTI systems and to implement useful operations on systems described this way. Section 4 of this handout contains examples of the operation of this class.

We will represent a system in terms of its system function, as a ratio of polynomials in $\mathcal{R}$. We can use a `SystemFunction` class with the following methods:

- `__init__(self, numeratorPoly, denominatorPoly)`: takes two instances of the `Polynomial` class as input and stores them in this `SystemFunction` instance as the attributes `numerator` and `denominator`.
- `poles(self)`: returns a list of the poles of the system. Remember that the poles are the roots of the polynomial in z, where $z = 1/\mathcal{R}$.
- `poleMagnitudes(self)`: returns a list of the magnitudes of the poles of the system. The magnitude of a real pole is simply its absolute value. The magnitude of a complex pole is the square root of the sum of the squares of its real and imaginary parts. The `abs` function in Python does the appropriate computation for both types.
- `dominantPole(self)`: returns a "dominant" pole, that is, one of the poles with the greatest magnitude (there may be more than one with the same, largest, magnitude; in this case, return any one of them).

Step 1. Edit `sfSkeleton.py` to contain your implementation of these methods. You can test it using `sl5Work.py`, which will load your `sfSkeleton.py`. We have set up `sl5Work.py` to import your definitions from `sfSkeleton.py` as `sf`, so that the examples match those in Section 4.

Hints and cautions

- To create a Polynomial, use `poly.Polynomial([...])`
- None of the operations that you implement should change any of their arguments. Be very careful of list operations that modify the input lists; e.g., `x.append`, `x.insert` and `x.reverse`.
- If you have a list bound to the variable `x`, then `x.reverse()` reverses the order of the elements of the list `x`. If you want to avoid affecting the original `x` you need to copy the list first, for example, by doing `y = x[:]`. Note that `y = x` does not copy the list, it simply creates a new name for the same list.
- You might want to use the procedure `util.argmax(l, f)`, which takes as input a list `l` and a procedure `f` that can take an element of `l` as input and return a numerical score as ouput. The result is the element of `l` for which `f` outputs the highest score.

Wk.5.1.1	Once you have debugged your code in Idle, paste it into this tutor problem, check it, and submit it.

3 Combining System Functions

We can also implement two basic operations for combining system functions, analogous to operations we saw for state machines. They are described in detail in Chapter 6 of the course readings.

Step 2.

Wk.5.1.2	Get practice with cascade combination of system functions.

Step 3. In `sfSkeleton.py`, implement the procedure `Cascade(sf1, sf2)`, which takes two instances of the `SystemFunction` class and returns a new instance of that class that represents the cascade composition of the input systems. *Although this is a procedure and not a class, we capitalize the name by analogy with the `sm.Cascade` class.*

Wk.5.1.3	Once you have debugged your code in Idle, paste it into this tutor problem, check it, and submit it.

Step 4.

Wk.5.1.4	Get practice with feedback-subtract combination of system functions.

Step 5. In `sfSkeleton.py`, implement the procedure `FeedbackSubtract(sf1, sf2)`, which takes two instances of the `SystemFunction` class and returns a new instance of that class that represents the feedback subtract composition of the input systems. *Although this is a procedure and not a class, we capitalize the name by analogy with the `sm.FeedbackSubtract` class.*

Wk.5.1.5	Once you have debugged your code in Idle, paste it into this tutor problem, check it, and submit it.

4 Examples

These examples, drawn from the notes, are included in `sl5Work.py`.

Real poles:

```
>>> s1 = sf.SystemFunction(poly.Polynomial([1]),
                           poly.Polynomial([0.63, -1.6, 1]))
>>> print s1
SF(1.000/0.630 R**2 + -1.600R + 1.000)
>>> s1.poles()
[0.90000000000000069, 0.69999999999999951]
>>> s1.poleMagnitudes()
[0.90000000000000069, 0.69999999999999951]
>>> s1.dominantPole()
0.90000000000000069
```

Complex poles:

```
>>> s2 = sf.SystemFunction(poly.Polynomial([1]),
                           poly.Polynomial([1.1, -1.9, 1]))
>>> print s2
SF(1.000/1.100 R**2 + -1.900R + 1.000)
>>> s2.poles()
[(0.94999999999999996+0.44440972086577957j), (0.94999999999999996-0.44440972086577957j)]
>>> s2.poleMagnitudes()
[1.0488088481701516, 1.0488088481701516]
>>> s2.dominantPole()
(0.94999999999999996+0.44440972086577957j)
```

Driving to a wall example from the notes:

```
>>> T = 0.1
>>> k = -2.0
>>> controller = sf.SystemFunction(poly.Polynomial([k]),
                                   poly.Polynomial([1]))
>>> print controller
SF(-2.000/1.000)
>>> plant = sf.SystemFunction(poly.Polynomial([-T, 0]),
                              poly.Polynomial([-1, 1]))
>>> print plant
SF(-0.100R/-1.000R + 1.000)
>>> controllerAndPlant = sf.Cascade(controller, plant)
>>> print controllerAndPlant
SF(0.200R/-1.000R + 1.000)
>>> wall = sf.FeedbackSubtract(controllerAndPlant)
>>> print wall
SF(0.200R/-0.800R + 1.000)
>>> wall.poles()
[0.80000000000000004]
```

SW Lab 5

10/5

- implement a LTI system
  - last weeks + this weeks lecture
- w/ SystemFunction class
- create polynomial w/ poly.Polynomial([ ])
- stuff should not modify list

X = reverse may be needed

Copy list w/ $y = X[:]$ ← so much easier!

argmax
- returns highest score as output
- the element which outputs highest value

So input $\frac{Y}{X} = \frac{1}{.63R^2 + -1.6R + 1} = \frac{N(R)}{D(R)}$

- oh same example as in lecture
  - makes it easier!

flip? → $\frac{1}{1 - 1.6R + .63R^2}$

Sub $R \to \frac{1}{z}$ - no don't need to flip

$\frac{1}{\frac{.63}{z^2} - \frac{1.6}{z} + 1}$ ← rooten to find denom

(2)

We already implemented cooten

- try it w/ denom as is now
- question is how to deal w/ $\frac{1}{z}$
- in the representation that is in class
- not what lecture got

- now its

$$.63\,z^2 - 1.6\,z + 1$$

we want

$$\frac{.6}{z^2} - \frac{1.6}{z} + 1$$

- ~~///~~
- $\frac{1}{n}$

- work w/

$$z^2 + \tfrac{1}{2}z - \tfrac{1}{8} \rightarrow \left(z + \tfrac{1}{2}\right)\left(z - \tfrac{1}{4}\right)$$

$$(z+1)(z-2)$$

$$z^2 - z - 2$$

So it works in isolation

~~- do that~~

- but still not what prof got in one

(3) Or is it reversed

[-.125, .25, 1] entered as

should be roots of

[1, .25, -.125].roots()

- I think that's what $\frac{1}{z}$ converts

- oh it copies lists
  - not polynomials
- just do it in init
- no - can't
- I solved this before
  - oh ~~it~~ will give coeffs depending on lib version
    polynomial
- came out right, but wrong sign
- multiply ans · -1
- no remember root as int is that

$(z+.5)(z-.25)$

$z=-.5$ $z=.25$ ∈ so correct

Now pole magnitudes

- simply absolute value if real
- imaginary $\sqrt{real+i}$ parts
- but abs does for both

(4)

- but abs does not do lists, grrr
  ok done

---

dominate pole

- return value of pole
- or position or what?

$[-.5, .25)$

$.5 > .25$

↑ return $-.5$

- ? or do I want to do argmax

util.argmax(poles, abs())

// // (self.poles, abs) ✓

---

should be done

tutor

worked on 1st try!

- all that's due today
- awake now, so work 40 more min

(5)

Cascade Practice

output input

$H_1 = w[n] - w[n-1] = x[n] - 2x[n-1]$

$H_2 - y[n] - y[n-1] = w[n] + w[n-2]$

oh do this 1st

H_1/ $w[n] = x[n] - 2x[n-1] + w[n-1]$

H_2/ $y[n] = w[n] + w[n-2] + y[n-1]$

H_1/ $Y = X - 2RX + YR$

H_2/ $Y = X + XR^2 + YR$

/ or is that right?

$H_1 = \frac{Y}{X} \Rightarrow$ $Y - YR = X - 2RX$

$Y(1-R) = X - 2RX$

$Y = \frac{X - 2RX}{1-R}$

$\frac{Y}{X} = \frac{1-2R}{1-R}$

(6)

H_2

$$Y - YR = X + XR^2$$

$$Y(1-R) = X + XR^2$$

$$Y = \frac{X + XR^2}{1-R} \approx \boxed{\frac{W + WR^2}{1-R}}$$

$$\frac{Y}{X} = \frac{1+R^2}{1-R}$$

H_1

$-2 \quad 1$ ✓

$-1 \quad 1$

H_2

$1 \quad 0 \quad 1$ ✓

$-1 \quad 1$

Now Cascade

- order does not matter.

do $H_1 \rightarrow H_2$ ← order confusing

$$Y = \frac{X - 2RX}{1-R}$$

$$\frac{\left(\frac{X-2Rx}{1-R}\right) + \left(\frac{X-2Rx}{1-R}\right)R2}{1-R}$$

? think that is it the output of 1 into other

(7)

$$\frac{Y}{X} = \frac{\left(\frac{1-2R}{1-R}\right) + \left(\frac{1-2R}{1-R}\right) R2}{1-R}$$

num $\frac{1-2R}{1-R} + \frac{R^2 - 2R^3}{~~R^2 - R^3~~}$ denom stays same $1-R$

? can you reduce it further

-duh - simple math screws me every time!

- or could you use the #

- find common denom

~~-RR~~ $(R^2 - R^3)$

So ~~$\left(\frac{1-2R}{1-R}\right)\frac{(R^2)}{(R^2)} \rightarrow \frac{R^2 - 2R^3}{R^2 - R^3} + \frac{R^2 - 2R^3}{R^2 - R^3}$~~

~~$\frac{2R^2 - 4R^3}{R^2 - R^3}$~~ ~~$(R^2 - R3)(2-4)$~~

$$\frac{1-2R+R^2-2R^3}{1-R}$$

~~does $\frac{2x-4y}{x-y} = 2-4$ factor!~~

~~$\frac{R^2(2-4R)}{R^2(1-R)} = \frac{2-4R}{1-R}$~~

(8)

together

$$\frac{\frac{2-4R}{1-R}}{\frac{1-R}{1}}$$

$$\frac{2-4R}{1-R} \cdot \frac{1-R}{1} =$$

$$\frac{2-4R}{1-R} \cdot \frac{1}{1-R} = \frac{2-4R}{R^2-2R+1}\ 1-2R+R^2-2R^3$$

Clear way

$= H_2 H_1 X$

$$y = \left(\frac{1+R^2}{1-R}\right) \cdot \left(\frac{1-2R}{1-R}\right) X$$

$$\frac{1+R^2-2R-2R^3}{1-2R+R^2} = \text{same thing much simpler}$$

$$-2 \quad 1 \quad -2 \quad 1$$

$$1 \quad -2 \quad 1$$

✓

Stop for today

10/5
night

(4) 5.1.3
implement cascade
- procedure • (SF1, SF2)
  τ τ
  2 sys Functions

$Y = M_1\ M_2\ X$

num= M_1 numerator $*M_2$ num
derom= M_2 derom M_2 deron

oh right not self. =
return a new SF
done ✓

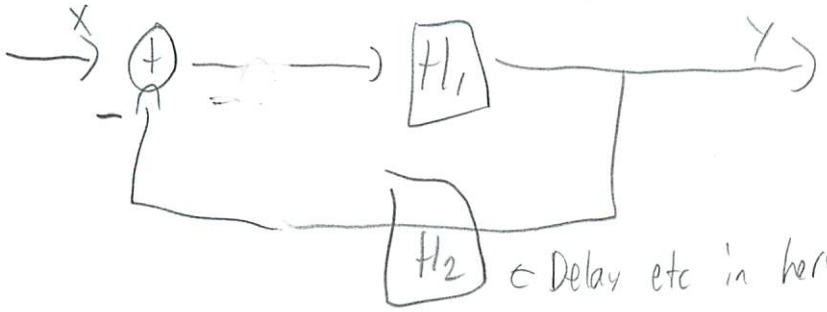
5.1.4 Feed back sub fract practice

$H_1 = \frac{N_1}{D_1}$ ← poly nomials in Q (like $\frac{Y}{X}$?)

$H_2 = \frac{N_2}{D_2}$

- must enter both together
- + x only
  2.N_1 = legal
  ( )

(10)



$Y = H_2[n-1] + H_1$

- but that is w/ r
- hmm, docs
  - formating is hard part
- ; in docs

feeding output of sf1 back w/ sf2 optionall

- no help

- oh here → $(N1 \circ D1) + N2$ ← example
- numerator is Y (output) = N
  X = input = D

num: $N1 + D1 = N2$

; denom what would that be?
- input

- look at lecture example

(11)

there it is

$$Y = H_1(x - H_2 Y)$$

$$Y + H_1 H_2 Y = H_1 X$$

$$Y(1 + H_1 H_2) = H_1 X$$

$$Y = \frac{H_1}{1 + H_1 H_2} X$$

$$\frac{Y}{X} = \frac{H_1}{1 + H_1 H_2}$$ ← Num ← Denom

but Hs $\frac{\frac{N_1}{D_1}}{1 + \frac{N_1}{D_1}\frac{N_2}{D_2}}$ ⓧ big no

- think how I did cascade
  - but that was different format
- oh no division allowed
- but could implement like that
  - and then see their ans
  - they say make sure fully simplified
- did in des lab 3 w/ Sm
  - but this is w/ Q functions
  - get next value

(12)

- before I had 2 examples

$$\frac{N_2 N_1}{D_2 D_1}$$ ← would have been my answer ← that is multiplication

- what is Black's formula
  - oh just $\frac{Y}{X} = \frac{H_1}{1 + H_1 H_2}$ ← part of
    - but what was my multiple nom/denom
    - should reduce

$$\frac{\frac{N_1}{D_1}}{1 + \frac{N_1}{D_1}\frac{N_2}{D_2}}$$

$$\frac{N_1}{D_1} \cdot \left(\frac{1}{1 + \frac{N_1}{D_1}\frac{D_2}{N_2}}\right)$$

So $$\frac{N_1 (D_1 \cdot D_2)}{D_1 (N_1 N_2)}$$ (X)

ti calc → enter and push enter

$$\frac{D_2 N_1}{D_2 D_1 + N_1 N_2}$$ (✓) bingo - how did they do it

- and I needed to see that lecture notes slide
- all the diff formats confuse me

(13)

Wolfram aka

$$\frac{N_1}{D_1\left(\frac{N_1}{D_2}+1\right)} \qquad \frac{D_2 N_1}{D_1(D_2+N_1)}$$

and the one I had

- oh well will ask about
- oh wait take what I had

$$\frac{N_1}{D_1}\left(1+\frac{D_1}{N_1}\,\frac{D_2}{N_2}\right)$$

can't just get rid of
why did I do that
I know that

$$\frac{N_1}{D_1}+\frac{N_1 D_1 D_2}{D_1 N_1 N_2}$$

$$\frac{N_1}{D_1}+\frac{D_2}{N_2}$$ < get common denoms

$$\frac{N_1 D_2}{D_1 D_2}+\frac{N_1 D_2}{N_1 N_2}$$ - nope

- hmm wait on it

~~Hint~~

$N_1 D_2$

(14)

Now implement ✓

Now pre-lab for tomorrow

- Construct wall finder system function
- Controller SF
  - takes gain - fun of error
  - returns velocity
- plant SF
  - T timstep duration
  - input velocity
  - output = actual distance to wall
- sensor SF
  - no in
  - out: R(sensor reading)
- wall finder system puts it all together
- "did we not do this before?
  - oh that was SM
- need to do w/ SF
- "so just change it"
  - yeah worked for controller!
- no initD
  - since doing R math

→ blank for wire

- is ans kinda in SLS work!

(15)

- well not exactly
- w/ math, not using PCAP Functions
- (V) worked
  - that would have made last Design lab much eaiser

Week 4 HW

10/6

- a bit late, but due tomorrow
  - well still on time
  - but after design lab

4.4. Finding systems

- oh more diff eq
- w/ d + c coeffics
- real life situations this time

$x[n]$ = digitized sound

$y[n]$ = avg of previous 2 inputs
forget first few samples

$$y[n] = (x[n-1] + x[n-2])/2$$

$$\frac{R + R^2}{2}$$

- hard # coefficient
- oh I remember

c			d
.5	.5	0 (X)	none (✓)
0	.5	.5 (X)	

- ? hmm

let me do others - only have 8 checks

(2)

What is it

input			output	
n	n-1	n-2	n-1	n-2
↑ starts different			↑	

on 0 , 5 , 5 ✓

- duh forgot format

Bank → 5% annual interest rate

inputs = deposits that year

output = balance at end of year

n = 1 year

- ~~only~~ * don't earn interest till year rolls over deposits

$y[n] = x[n] + 1.05\,y[n-1]$

1 ✓	1.05 ✓

- you deposit $100 in n=0
- what is balance in years
  - hint write a program
  - ; w/ our thing sf?

~~$y[n] = x[n] + A$~~

$y[n] = x[n] + 1.05\,y[n-1]$

$Y = X + 1.05\,A\,y$

(3)

$y - 1.05 R_y = x$

$y(1 - 1.05R) = x$

$\frac{y}{x} * (-.05R) = 1$

$\frac{y}{x} = \frac{1}{-.05R}$

SF (Polynomial[1], Polynomial[-.05, 0])

- ? so how eval that ?
  - no step ?
- a for loop would have been so much simpler
- can you plot it ?
  - wrong thing
- So in 2 min I made a non P-Cap loop
- guess I could have done a SM
  - but too complex to think it all through

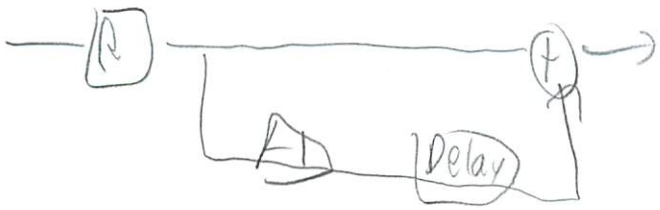
4.4.2 Representations

- So same dcoeff ccoeff
- but reading a picture
- Operator equation

$a_0 y + a_1 R_y + a_2 R2_y$

Constant (R0) ← yeah, so ?

(4)



- this could have helped w/ other one

$$y[n] = x[n-1] + -1\,x[n-2]$$

d	c
0 1 -1 ✓	none ✓

oh operator eq is something different

R_x	R_y — what is difference?
0 1 ✗	0 0 -1 ✗

- no it is just other form
  - according to past 6.01 notes

$$Y = XR - XR^2$$

0 1 -1 ✓	none ✗ — or something? — guess not

because it is = to 1

(5)

$$XR - xR^2 = Y$$

↑

1 ✓

Now much more complex pic

- first part is ~~same~~ similar

$Y[n] = x[n] + -1x[n-1] + 2x[$ all that! $)$

- hmm how do we do it

- look in lecture notes
- assign variables halfway

$W = X - 1xR$

$Y[n] = -1RRW + 2RW$

now plug in

$Y = -R^2(x - XR) + 2R(x - xR)$ ← guess this form exist here

$Y = -R^2x + XR^3 + 2Rx - 2R^2X$

$XR3 - 3R^2x + 2Rx$

1	-3	2	0 ⊗	none	✓
1	-3	2	0 ⊗	1	✓

Seems right

(6)

~~10/6~~

Oh did I do coefficients wrong
- diff order than polynomial class
- made same mistake as before

0 2 -3 1 ✓
0 2 -3 1 ✓

4.4.3 LTI SM

10/6

- built complex SM out of primitives
- guy told me this is how can do that bank thing
- write LTISM(d Coeff, c Coeff, previous In, previous Out) function
  - ?
  - ↑ ? why need both
    they seemed to be same format!
- oh don't need those, all the time
  - * last part?
  - why not ?
  - seems like you never need it

$y[n] = 3\,x[n]$

LTISM([3], [ ])

(7)

Delay Machine

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

LTISM([0,1], [ ], [0], [ ])

↑ oh specified $x[-1] = 0$

So like init

Oh!!! this is different than previous questions

- Ok now I get it

- try designing some

- n-1

$y[n] = \cancel{x[n-1]} + y[n-1]$

oh skip most recent input

none	1	0	10
↑	✓	✓	✓

Can't be

- would never increment

I has a right

0 1

- now scaled by .1

0	.1	1	8	1

4.4.4 Implement LTISM

- first simulate

LTISM([1,2],[1],[3],[4])

$y[n] = x[n] + 2x[n-1] + y[n-1]$

$x[-1] = 3$

$y[-1] = 4$

transduce [1, 2, 3, 4, 5]
(0, 1, 2, 3, 4)

$0 = n \Rightarrow 1 + 2 \cdot 3 + 4 = 11$ ✓

$1 = n \Rightarrow 2 + 2 \cdot 1 + 11 = 15$ ✓

$2 = n \Rightarrow 3 + 2 \cdot 2 + 15 = 22$ ✓

$3 = n \Rightarrow 4 + 2 \cdot 3 + 22 = 32$ ✓

$4 = n \Rightarrow 5 + 2 \cdot 4 + 32 = 45$ ✓

adds the previous 2 to accumulator

- but how does R help?

---

now code it up

- back to SMs
- implement get next value
- but how to process coefficents?

(9)

- remember dot product function
- think thru cases w/ d/c coeff empty
- so remember sm thinking
  - we did something like this already
  - w/ polynomial
    - poly R
    - but that is signal

- So # what to do
  - see I forget a lot of the sm stuff
- d coeff
  - well polynomial does that
- so lets use example to think through
  - go through d coeff adding
  - for x in d coeffic
    - x.input + ans = ans
    - don't change anything!
  - For x in coeff
    - x.getNextValue( ) + ans = ans
    - or will this be in state? may need to test in idle
  - but w/ inital #
    - if need n-1
  - or what if I did something crazy
    - output(n) & regens everytime (stateless though)
    - input(n)

(10)

So their start state would be

(3, 4)

- what do they intend state to be?
  - should use it, since that is point of class
- so ~~(n-1, n-2)~~ (x[n-1], y[n-1]) - but that is not flexible
- or (x[n-1], ~~x[n-2], x[n-3]~~)
  - but inflexible:

what if ([x[n-1], x[n-2], y[n-1], y[n-2])

which is what it would be

- so state always carries ~~n~~ this
- if no n-1 terms won't be used
- if not enough init terms, then program would error

So do as state

and forget output(n)
input(n)

- So get next value can only be done incrementally
  - So can't do in the loop

(11)

So get next value

```
ans = 0
i = 0
for x in d coeff
    x * input[i] + ans = ans
    i ++
```

↑ or this will be ~~switch on i~~

if i = 0 → the input

if i > 1 → state[0][i-1]

(will have to reform)
- put outside loop

```
i = 0
for x in coeff
    x * state[1][i] + ans = ans
    i ++
return (ans, state)
```

↑ or is it other way around?
- also need to change state
- return state, ans

- new state
  (input, ans)
  ? but what if more than one pre specified then the whole thing will be blanked on first run

- need to append these at start of list
- and remove old values
  ? really? - no! - no need (buffer overflow?)

- I get it, I just need a long time to think it out. I need to look back at the documentation

(12)

- well use insert (0, x)
- ok give it a try
  - ✓ worked 1st try!!!
    - never tested once
    - thought through every theoritical possibility

---

try the bank problem (my own personal try)

LTISM [[1], [1], [0], [0]) . transduce [[100, 0, 0, 0...]]

1 ← .05

0 ← need or willerror

no gets ∞ smaller

do again

~~y[n] = x[n]~~

Oh duh 1.05

woot! got it fairly easily!

4.4.5 TransducedSignal

- given a SM + sig we can construct new signal
  - but only works for $n \geq 0$
- So ~~better~~ this is what I wanted to do before ... or something
  - but we were only doing gains from unit signals like that

TransducedSignal (Signal) ← subclass

- __init__(self, S, M)
  - S: input signal
  - M: SM will use

- ~~An~~ sample(n)
  - returns value at n
  - start state machine from scratch to that value
    - so this is like my input()/output() function idea
    - well a total rejection of it
    - uses SM
      - not breaking model
- inefficient
  - can cache for "bonus points"
    - no actual bonus points
- ok so need to do sample for n > 0

  SM.transduce(signal)
  - but only to n
  - and return that value
- need write own test cases
  - darn!
  - find a signal and a SM

(14)

- Can't transduce a ~~Signal~~ SM?

  - well can but for value it must be #?
  - not a Signal instance
  - signal.sample(n)

    ↑ but need n up to it ;(

So

need ~~for~~ while $i \geq n$

  stream.append(s.sample(i))

and then m.transduce(stream)

- I spend a lot of time to do well in this class
- will see how tests are

↻ loop

- oh did not $i = i + 1$
- and $\leq$ not $\geq$

- and self.s
- ok now I need to ~~capture~~ only return last value, not whole transcript up to that pt
- actually strike that!

  - just run last one!

(15)

So no m.transduce(steam)
m.step(stream)
??
try it out
- they are using ltism as a SM
- hmm why is it erroring w/ no numpy?
- grrr
- fixed, took like 15 min!
- nope not stop
kinda doing G+V - not concentrating
- hmm just return last value of the transduce?
- ~~#~~ hacky
- pop!
✓ works

Now one more bank qu
- need an input signal
- ? anything?
- LTISM of bank
- construct signal by transducing signal through sm

(16)

- could just do LTISM

$$y[n] = .99 y[n-1] + x[n]$$

Or can use signal

100 gain 0
20
50

- Use poly R
  - must be better way!
- use SM I did last time _ actISM
- hmm their ltism will error if you give it to many inputs!
- ok got it
- now can I ans
- oh before it was since it was doing samples in range
  - still should have just reported out put

✓ all done

hand 5

10/7

- machine

$\frac{Y}{X}$ = R = n-1

- roots of denom ✓
- but

$$y[n] = (1+a)\,y[n-1] + (1+b)\,x[n] + b\,x[n-1]$$

$$Y - (1+a)_y R = (1+b)_x + bR_x$$

$$Y(1-(1+a)R)$$

$$\frac{Y}{X} = \frac{bR + (1+b)}{1 - 1 - aR}$$

$(x-1)$

roots $a+1$, none

got 12/13

Design Lab 5: Staggering Proportions

1 Introduction

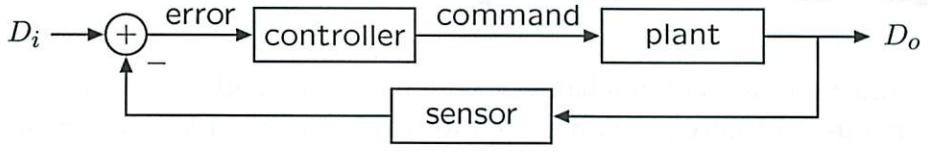
This lab should be done with a partner. Each partnership should have a lab laptop or a personal laptop that reliably runs soar. Do `athrun 6.01 update` to get the files for this lab, which will be in `Desktop/6.01/lab5/designLab/`, or get the software distribution from the course web page.

The relevant files in the distribution are:

- `propWallFollowBrainSkeleton.py`: A brain with a place for you to write the proportional controller.
- `wallTestWorld.py`: A world with a wall for the robot to follow. (This file appears in the `worlds` subdirectory)
- `dl5Work.py`: A file with appropriate imports for making system functions and finding their properties.

Be sure to mail all of your code and plots to your partner. Each of you will need to bring copies with you to your first interview.

In this lab, we will program the robot to move along a wall to its side, maintaining a constant, desired distance from the wall. We will use a simple *proportional controller*, and model it as a system with the same structure that we used for last week's *wall finder* system.



The steps in this lab are:

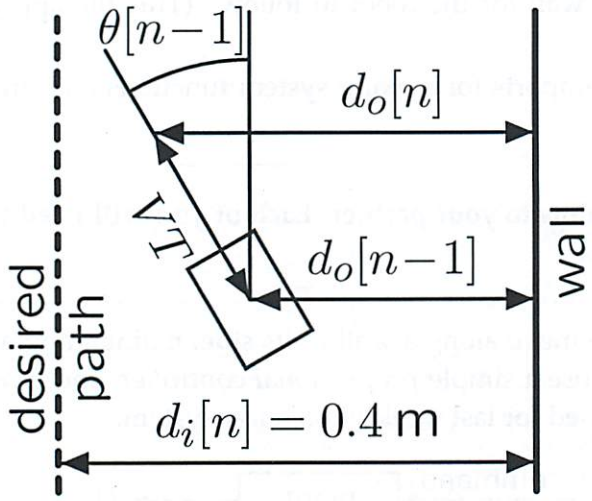
- Build a proportional controller for a robot and test it in simulation with different gains.
- Build an analytical model of the controller-plant-sensor system, both by hand and using the `SystemFunction` class.
- Use the model to gain understanding about the best gain to use for the controller and what kind of behavior to expect from the system.

Definitions of symbols

Here are the definitions of various symbols we will use in this lab. The descriptions might not yet make sense to you, but we put them here so you can refer back to them as you work.

- k : gain of the controller, a constant number
- V : forward velocity of the robot, a constant number
- T : the amount of time between discrete samples of the signals, a constant number
- D_i : desired distance of the robot to the wall, a signal whose samples are $d_i[n]$
- D_o : actual distance of the robot to the wall, a signal whose samples are $d_o[n]$
- E: error, equal to $D_i - D_o$, a signal whose samples are $e[n]$
- Θ : robot's angle with respect to the wall, a signal whose samples are $\theta[n]$
- Ω : robot's angular velocity, a signal whose samples are $\omega[n]$

2 Proportional wall-follower



The figure above illustrates a robot in a hallway, with its desired path a fixed distance from the right wall. We can build a controller with a fixed forward velocity of $V = 0.1\,\text{m/s}$, and adjust the rotational velocity $\omega[n]$ (not shown) to be proportional to the *error*, which is the difference between the desired distance $d_i[n] = 0.4\,\text{m}$ to the right wall and the actual distance $d_o[n]$. The constant of proportionality between the error and the rotational velocity is called the *gain*, and we will write it as k. Notice that when the rotational velocity $\omega[n]$ of the robot is positive the robot turns towards the left, thus increasing its angle $\theta[n]$.

Look in the file `propWallFollowBrainSkeleton.py`. The brain has two state machines connected in cascade. The first component is an instance of the `Sensor` class, which implements a state machine whose input is of type `io.SensorInput` and whose output is the perpendicular distance to the wall on the right. The perpendicular distance is calculated by `getDistanceRight`

in the `sonarDist` module by using triangulation (assuming the wall is locally straight). All of the code for the `Sensor` class is provided.

The second component of the brain is an instance of the `WallFollower` class. Your job is to provide code so that the `WallFollower` class implements a proportional controller.

Check Yourself 1. What should be the types of the input to and the output from a state machine of the `WallFollower` class?

Step 1. Implement the proportional controller by editing the brain. Then use soar to run your brain in the world `wallTestWorld.py`. The brain is set up to issue a command during the setup to rotate the robot, so it starts at a small angle with respect to the wall.

Step 2. Experiment with a few different values of the gain `k` of the controller (within the range 0 to 20). Generate slime trails to illustrate how the system's performance depends on the value of `k`; how does it affect the convergence of the output to the desired value? **Be sure to take and save screen shots of at least three slime trails.**

Checkoff 1.
- Show your slime trails to a staff member, and explain the implications of the slime trails for choosing a good value of `k`.
- Compare the kinds of behaviors exhibited by the wall-follower system to the range of behaviors of the wall-finder system from last week.

3 Mathematical model

Picking gains and trying them on the robot can become tedious, so we will build a model of the wall-follower system, and use it to understand, analytically, how the performance of the system depends on the gain. (Section 6.5 of the readings illustrates a similar modeling effort for the wall-finder system).

Step 3. **Controller model:** Assume that the controller can instantly set the rotational velocity $\omega[n]$ to be proportional to the error $e[n]$. Express this relation as a difference equation.

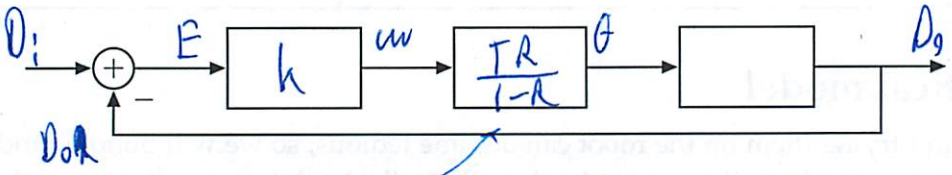
Step 4. **Plant model:** Write difference equations describing the "plant," that is, an expression for $d_o[n]$, that depends on ω. It is useful to break this problem into two parts:

- **Plant1:** Write an expression for $\theta[n]$, the robot's angular orientation with respect to the wall, that depends on ω. (Remember that the robot (or state machine) can only set new values for variables based on values that were computed at previous times, or in other words, each of the state machines in a brain update their variables in synchrony.)

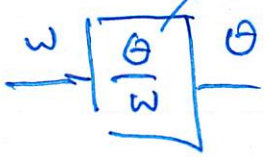
- **Plant2:** Write an expression for $d_o[n]$, that depends on θ, by linearizing the relation between d_o and θ using the small angle approximation (i.e., if θ is small, then $\sin\theta \approx \theta$). This approximation makes our model linear, allowing us to analyze it easily. It is useful to think about what its consequences are, for large angles, however.

Sensor model: To keep things simple, we will model the sensor as a wire: that is, assume it introduces no delay.

The subsystems represented by the three difference equations above connect together to form a system of the following form. Label each wire in the block diagram with the name of the corresponding signal. You may also find it useful to label each of the boxes in this diagram with the element of the model (controller, plant1, plant2) corresponding to each of the three difference equations you derived.



Step 5. Convert your difference equations into operator ($\mathcal{R}$) equations, and find the system function. If you're not sure about your difference equations, you might want to check with a staff member before solving.



$\theta = \theta R + \omega RT.$

$\theta - \theta R = \omega RT$

$\theta = \frac{\omega RT}{(1-R)}.$

$$H = \frac{D_o}{D_i} =$$

Wk.5.3.2 Enter the system function into the tutor by entering the numerator and denominator polynomials.

4 Software model

Manipulating equations, as you did above, is useful because you can keep constants like k and T as symbols, and see how they factor into the model. However, manual manipulation of equations can be quite error prone. In this section, we will explore the use of our Python `SystemFunction` class to model and analyze the wall-follower. The only drawback is that we will only be able to ask it to do so for particular numeric values of k and T.

The `SystemFunction` class provides two primitive kinds of system functions: gain (`sf.Gain`) and delay (`sf.R`). They are implemented as Python procedures, but named with uppercase variables by analogy with the `sm.Gain` and `sm.R` state machines.

```
def Gain(k):
    return sf.SystemFunction(poly.Polynomial([k]), poly.Polynomial([1]))
def R():
    return sf.SystemFunction(poly.Polynomial([1, 0]), poly.Polynomial([1]))
```

These can be combined, using `sf.Cascade`, `sf.FeedbackSubtract`, and `sf.FeedbackAdd` to make any possible system function; and the structure of the combination will be the same as it would have been to build up the analogous state machine.

Note that while the internal definitions of `Gain` and `R` use polynomials to construct them, by abstraction you can use these system functions, together with our means of combination without ever having to utilize these internal details.

Check Yourself 2. Use gains, delays, and adders to draw system diagrams representing the controller and each of the parts of the plant you derived in the previous section.

Step 6. Edit the `dl5work.py` file to implement Python procedures called `controller`, `plant1`, and `plant2` that use `sf.Gain`, `sf.R`, `sf.Cascade`, `sf.FeedbackSubtract`, and `sf.FeedbackAdd` to construct and return instances of the `SystemFunction` class that represent the three blocks in the mathematical model. Pass the important parameters for each block (e.g., `k`) as inputs to the corresponding procedure.

Step 7. Write a Python procedure `wallFollowerModel(k, T, V)` that calls the previous Python procedures to make system functions for the components and composes them into a single `SystemFunction` that describes the system with `desiredRight` as input and `distanceRight` as output.

Wk.5.3.3 Enter the definition for `wallFollowerModel` and any procedures it calls into the tutor. Do not enter any `import` statements.

Checkoff 2. Demonstrate to a staff member that, for $k = 1$, $T = 0.1$, and $V = 0.1$, the system function returned by your model is the same as the one you derived mathematically.

5 Model predictions

Now, we will use our models to understand what the system can and cannot do. Let us assume throughout this section that $T = 0.1$ and $V = 0.1$. These are reasonable values for our robots.

We know how to make predictions about how a system, starting at rest, will respond to a unit sample signal as input. Our situation here is different, in that the system does not start at rest and that the input is a persistent step signal (asking the robot to achieve a fixed distance from the wall). We will see next week that the unit-sample response of the system at rest is nonetheless an important predictor of behavior in the more general case.

Step 8. Use the `dominantPole` method of the `SystemFunction` class, to determine what your model predicts, for each of the gains 0.5, 1.0, 2.0, 4.0, and 10.0. In response to the unit sample signal as input,

- Will the output signal oscillate?
- Will the output signal converge to 0?
- If it does converge to 0, how many time steps would it take for its magnitude to go below 0.01? (In Python, `math.log(v, b)` computes $\log_b v$.)

Step 9. Find an algebraic expression for the poles of the system. Be sure it agrees with the software for gains 0.5, 1.0, and 10.0.

Step 10. Find the range of gains, if any, that will make the system stable (have bounded output in response to bounded input).

Checkoff 3.

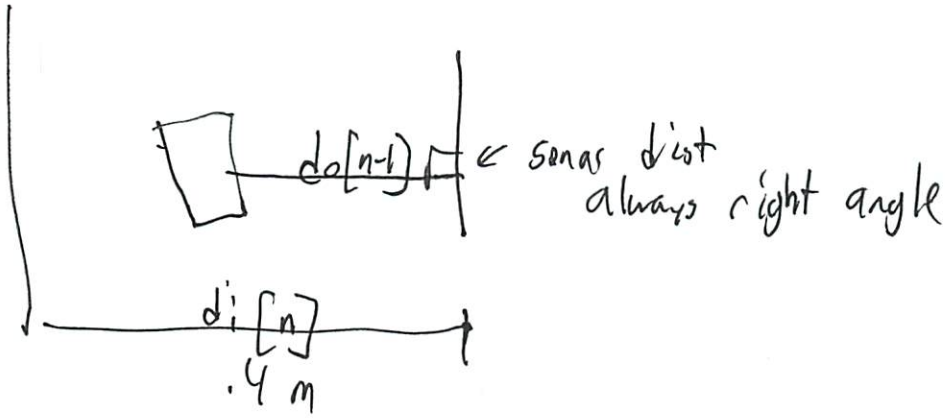
- Explain how you found the range of stable gains in the last step. Discuss how they relate to the behavior observed in the robot simulations from **section 2.**
- Explain what can you do with a state machine representation of a system that you can't do with a system function representation of that same system? What can you do with a system function representation that you can't do with a state machine representation?

Des Lab 5

10/7

- Wall follower, get next value
  - inp = distance
  - out = ~~θ~~ ?, commands ?
    - yeah w, θ action
      - V = .1 θ = ?

need to calculate θ



do[n-1] = sensor = < .4 θ ∈ ~~θ~~ - ?
↑ proportional

> .~~5~~4 θ ~~=~~ .1

k the greater the distance from .4, the greater the ~~θ~~ error
- try constants

error = ~~abs(~~ actual - D_i)

theta = k * error
↑ experiment

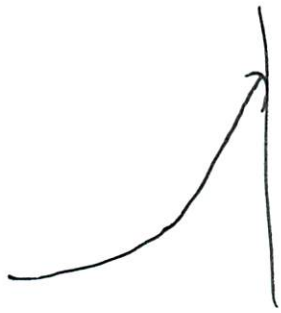
(2)

Will it ossilate

- depends on gain

Desired angle relative to ~~robot~~ robot;
or compass like

'i will 'it ossilate



'i wrong gain

- but very different in wall test world

k= .1, .5, 1 hits

k=10 swings widly, does a loop

k=5 swings are still kinda far

'i did 10 swing smaller

k=20 kinda nice

k= 208 just does circles

* key we are not changing vel - could get it to work better by specifing that

③ Less overshooting

Checkoff 1 ✓

Controller

$$w[n] = k\left(D_i[n] - D_o[n-1]\right)$$

Plant

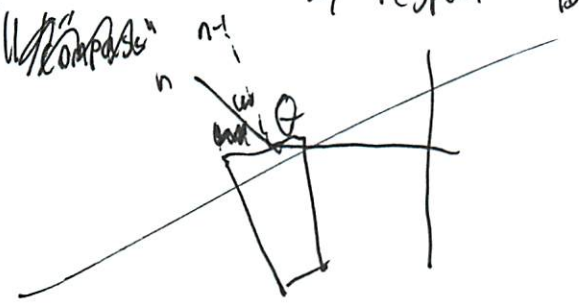
$$d_o[n] = f(w)$$

Plant 1

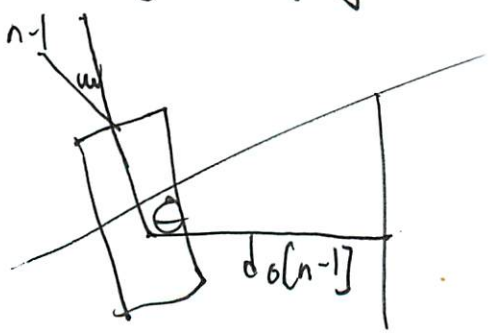
$$\theta[n] = f(w)$$

r — c w/ respect to robot

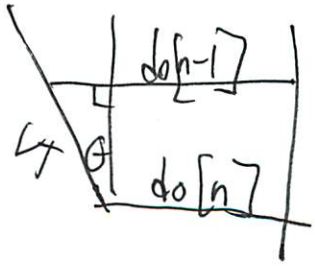
w/ respect



$$\theta[n] = w[n-1] - w[n]$$



look at pic



$$d_o[n-1] - d_o[n]$$

$$\sin\theta = \frac{d_o[n-1] - d_o[n]}{v[n] \cdot t}$$

$$\theta = \sin^{-1}\left(\frac{d_o[n-1] - d_o[n]}{v[n] \cdot t}\right)$$

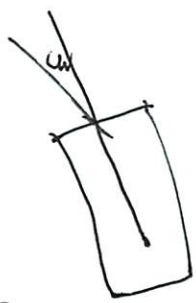
(4)

Opps we did plant 2

TA: Use approx $\sin\theta = \theta$

so $\theta[n] = \dfrac{d_o[n-1] - d_o[n]}{V[n] \cdot T}$

Plant 1



$\theta[n] = \theta(n-1) + w[n-1]T$

Plant 2 clarification

$d_o[n] = f(\theta)$

reverse what we did before

$d_o[n] =$

$\sin\theta = \dfrac{d_o[n] - d_o[n-1]}{V[n-1]T}$

$\theta \quad \sin\theta \, V[n-1]T + d_o[n-1]$

$d_o[n] = \sin\theta \, V[n-1]T + d_o[n-1]$

fill in for θ

$d_o[n] = \underbrace{\left(\theta[n-2] + w[n-2]T\right)}_{\theta[n-1]} \overbrace{V[n-1]}^{\text{Constant}} T + d_o[n-1]$

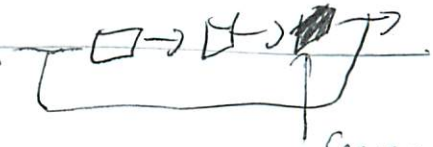
(4a)

$$d_0[n] = \theta[n-1]\, VT + d_0[n-1]$$

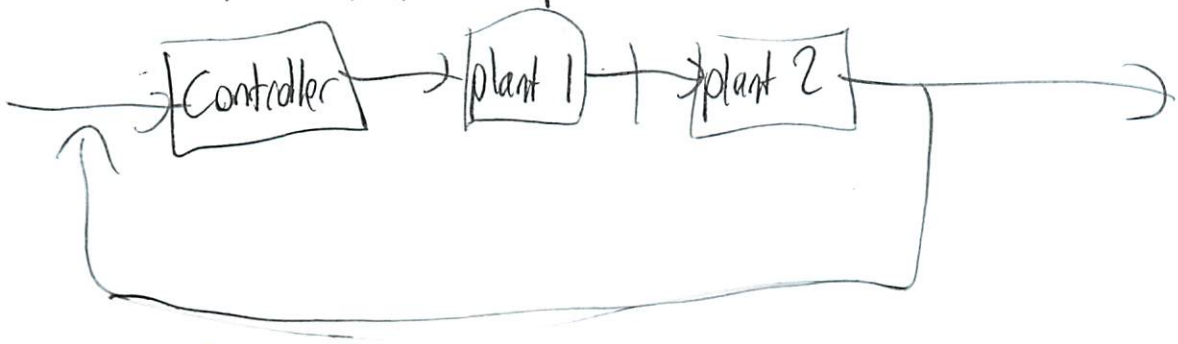
(5)

Sensor

But how can you do wire, won't lead to ∞ loop?

~~Oh not~~ Feed ~~back~~ back in this 

No that is plant 1, 2



Convert to R

Rewrite $u[n] = k(D_i[n] - D_o[n-1])$

$\theta[n] = \theta[n-1] + u[n-1]\,T$

$d_o[n] = \theta[n-1]\,VT + d_o[n-1]$

$d_o[n] = \theta[n-2] + u[n-1]\,T^2V + d_o[n-1]$

$= \theta[n-2] + k[D_i[n-1] - D_o[n-2]]\,T^2V + d_o[n-1]$

$\rightarrow (\theta R^2 + k(D_i R - D_o R^2))T^2V + d_o R$

$= VT^2\theta R^2 + kT^2V D_i R - VT^2 k D_o R^2 + d_o R$

should do multiply (cascade) + feed back subtract pattern

$\frac{Y}{X}$ thing ↑ more confusing

let me try my way

(6)

~~each part as~~

$$d_o - d_o R = VT^2\theta R^2 + kT^2 V D_i R - VT^2 k D_o R^2$$

$$d_o - d_o R + VT^2 k D_o R^2 = VT^2\theta R^2 + kT^2 V D_i R$$

$$d_o (1 - R + VT^2 k R^2) = \text{"}$$

$$\frac{d_o}{d_i} \frac{Y}{X} = \frac{VT^2\theta R^2 + kT^2 VR}{1 - R + VT^2 k R^2}$$

- can not answer in terms of θ
- but our def of θ refers to θ

$$\theta = \sin^{-1}\left(\frac{d_o[n-1] - d_o[n]}{VT}\right)$$

$$\approx \frac{d_o R - d_o}{VT}$$

$$d_o - d_o R + VT^2 k D_o R^2 = VT^2\left(\frac{d_o R - d_o}{VT}\right) + kT^2 V D_i R$$

$$d_o RT - d_o T$$

$$d_o + d_o T - d_o R - d_o RT + VT^2 k D_o R^2 = kT^2 V D_i R$$

(6.b)

$$d_0\left(1+T-R-RT+VT^2kR^2\right) = kT^2V_{bi}A$$

$$\frac{d_0}{di} = \frac{kT^2VR}{VT^2kR^2-R-RT+1+T}$$ (X) (X)

(7)

TA: Got everything in Q

$$w = k(D_i - D_o R)$$

$$\theta = \theta R + \mu T R = \frac{d_o R - d_o}{VT}$$ what does it = — need to get θ out

$$d_o = \theta R V T + d_o R$$

$$d_o = \left(\frac{d_o R - d_o}{VT}\right) RVT + d_o R$$

$$d_o R^2 - d_o R + d_o R$$

→ when do d_i — not enough

$$d_o = (\theta R + \mu T) RVT + d_o R$$

$$(\theta R + (k(D_i - D_o R)T))RVT + d_o R$$

$$\left(\left(\frac{d_o R - d_o}{VT}\right)R + kTD_i - kTD_o R\right) RVT + d_o R$$

$$\frac{VT d_o R^2 - d_o R^2 VT}{VT} + kT^2 V D_i R - kT^2 D_o R^2 + d_o R$$

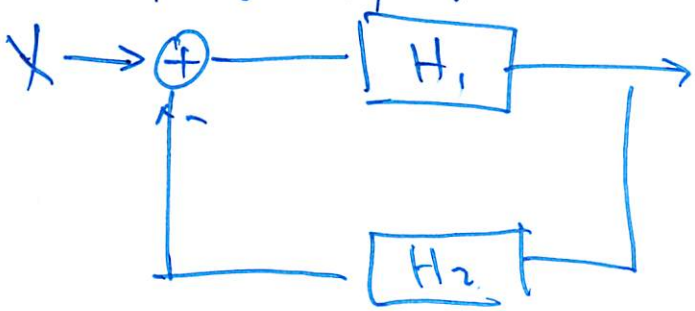
$$-d_o R^3 + d_o R^2 + kT^2 D_o R^2 - d_o R = kT^2 V D_i R$$

$$d_o(-R^3 + R^2 + kT^2 R^2 - R) = kT^2 V D_i$$

$$\frac{d_o}{d_i} = \frac{kT^2 VR}{-R^3 + R^2 + kT^2 R^2 - R}$$ ← same ← different

(8)

Prof: Try again w/ System Functions for each



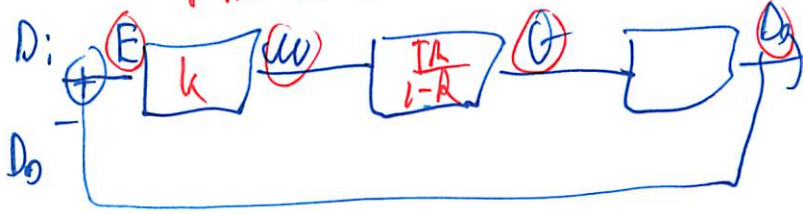
$$\frac{Y}{X} = \frac{H_1}{1 + H_1 H_2}$$

wire $H_2 = 1$

H_1 = cascade → multiply

$\omega = kE$

Fill in what have at each pt



1st

$$\frac{\omega}{E} = k$$

2nd

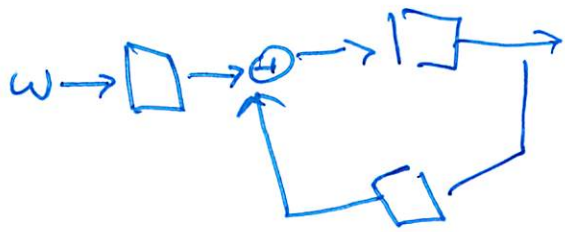
$$\frac{\theta}{\omega} =$$

$\theta = \theta R + \omega TR$

$\theta - \theta R = \omega TR$

$\theta(1-R) = \omega TR$

$$\frac{\theta}{\omega} = \frac{TR}{1-R}$$



10/7

core concepts

Review of Formats

→ Diff eq

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

SM

→ R

$$Y = X + XR + YR$$

"operator equation"

Rx coeff: 1, 1

Ry coeff: 1, 1

~~polynomials~~

~~[Rx coeff, Ry coeff]~~

d coeff: [1, 1]
↑
R^0

c coeff: [1, 1]

no memorize will show tutor/code rep

→ System functions

$$\frac{Y}{X} \#$$

$$Y - YR = X + XR$$

$$Y(1-R) = X + XR$$

$$\frac{Y}{X} = \frac{1+R}{1-R}$$

polynomials in R

$$\frac{[1,1]}{[1,1]}$$
↑
R^0

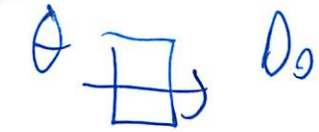
LTISM

~~c coef~~ - SM

- or a transduced signal

(9)

Box 3



$$d_0 = d_0 R + VTR\theta$$

$$d_0 - d_0 R = VTR\theta$$

$$\frac{d_0}{\theta} = \frac{VTR}{1-R}$$

Ratio

T' include

Together

multiply → cascade

$$k \cdot \frac{TR}{1-R} \cdot \frac{VTR}{1-R}$$

$$\frac{d_2}{d_i} = \frac{k \cdot T^2 R^2 V}{(1-R)^2}$$

Now plugs formula for feedback

$H_1 =$

$H_2 = 1$ (wire)

(10)

$$\frac{\frac{kT^2R^2V}{(1-R)^2}}{1 + \frac{kT^2R^2V}{(1-R)^2} \cdot 1}$$

simplify

Multiply all terms by $(1-R)^2$

$$\frac{kT^2R^2V}{(1-R)^2 + kT^2R^2V}$$

try entering
- don't need to simply more

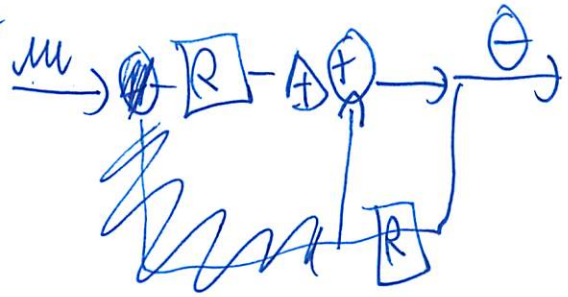
Now write code
- what does it return
- System Function (Numerator, Denom)

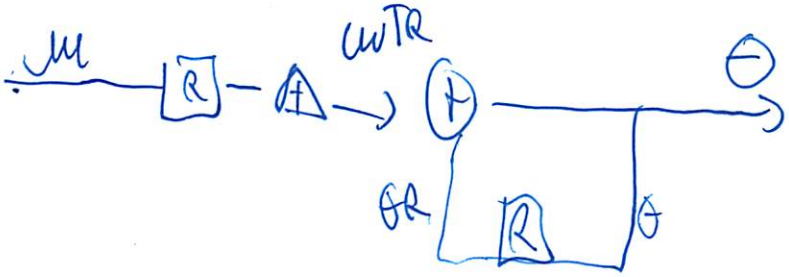
```
sf.Cascade(sf.Gain(T), sf.R)
```

- draw out blocks for each part
- go back to system function

(11)

plant 1



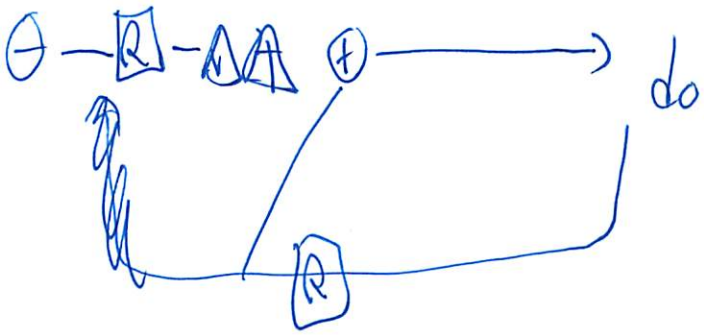


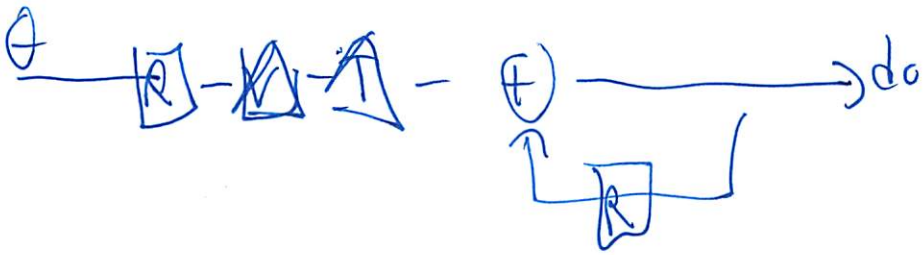
✓

How to do a Feedback add w/ wire? !?

Sf. gain (1)

plant 2





together

~~Feedba~~ Cascade Controller, plant 1, plant 2

feedback add (✓ , gain (1))

↑ wire (can leave blank

(12)

So I get

$SF(.001R^2 / 1.001R^2 - 2R + 1$

SF → $\dfrac{.001R^2}{1.001R^2 - 2R + 1}$

$k = 1$
$T = .1$
$V = .1$

we had $\dfrac{kT^2R^2V}{(1-R)^2 + kT^2R^2V}$ plug + chug

$$\frac{(1)(.1)^2(R^2)(.1)}{(1-R)^2 + (1)(.1)^2R^2(.1)} = \frac{.001R^2}{(1-R)^2 + .001R^2}$$

$R^2 - 2R + 1$

$\dfrac{.001R^2}{1.001R^2 - 2R + 1}$ ✓ same

Check off 2 ✓

done for today

(13) Model Predictions

10/7 home

Understand what system will do + not

$T = .1$ $V = .1$ constants

How will it respond to a sample signal?

? Confused something about how we can't predict system state but we will do anyway?

Use dominate pole method w/ different gains

.5	$1 + .0223J$
1	$1 + .0316J$
5	$1 + .0707J$
10	$1 + .1J$
20	$1 + .1414J$

- will ossilate, converge to 0?
- if converges to 0 → how many time steps for magnitude to $\leq .01$
  - $-\log_b V$ = math.log(v, b)

- I think I am doing this wrong
- like deslab 4
  - but that was a transduced signal
- where were poles introduced SL 5?
  - but did we ever test

(14)

- explained in lecture 5 slides

>1	unbounded
≤ 1	bounded / stable
~~1~~	transient → will converge to 0

- description of input
- non-transient = persistant

- but what about j ?
- or howabout trying - values
  - last time

real	>0 / ⊕	increase / decrease monotonically
real	<0 / ⊖	~~decrease~~ alternate signs
Complex	- periodic	

So when gain $=0$ → pole $=1$ → bounded

gain <0 → pole >1 real → monotonically unbounded

gain $\neq 0$ → ~~Complex~~

pole $\neq 1$ Complex → periodic

ask in office hrs when go for chella oft

(15)

All Linear time invariant

Checkoff 3

L0/11 Office Hrs

	SF	SM	LIISM
values	X short term/ transient	✓	✓
Convergence	✓ long term/ persistent	X Can do it, no automatic way won't be asked to do it don't know when it becomes persistent	✓ had 4 arguments
info	numerator (poly) denom (poly)	string of python	num denom init x init y

(16)

Solve for poles

$$\frac{k T^2 R^2 V}{(1-R)^2 + k T^2 R^2 V}$$

~~$\frac{k(.01) R^2 (1)}{(}$~~ did on pg 12

~~keep~~

$$\frac{.001 R^2}{1.001 R^2 - 2R + 1}$$ ← enter this $[1.001, -2, 1]$

$R = \frac{1}{z}$

$$\frac{.001\left(\frac{1}{z}\right)^2}{1.001\left(\frac{1}{z}\right)^2 - 2\left(\frac{1}{z}\right) + 1}$$

keep k

$$\frac{.001 \frac{1}{z^2}}{1.001\left(\frac{1}{z^2}\right) - 2\frac{1}{z} + 1}$$

want polynomial in z

so $\cdot z^2$

$\left(\frac{1}{z^2} \cdot z^2 = 1\right)$

(17)

$$\frac{.001 \cdot 1}{1.001 \cdot 1 - 2z + 1z^2}$$

$$\frac{.001 \cdot}{1z^2 - 2z + 1.001}$$

$a \quad b \quad c$

$[1, -2, 1.001]$

polynomial in z

never enter!

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

correct

$$\frac{2 \pm \sqrt{4 - 4 \cdot 1 \cdot 1.001}}{2a}$$

did not keep k

$$\frac{2 \pm \sqrt{-.004}}{}$$

---

start over w/ k

$$\frac{k(.1)^2(R^2)(.1)}{(1-R)^2 + k(.1)^2 R^2 (.1)}$$

$$\frac{.001 k R^2}{(1-R)^2 + .001 k R^2}$$

(18)

$$\frac{.001kR^2}{(R^2-2R+1)+.001kR^2}$$

$$\frac{.001kR^2}{(1+.001k)R^2-2R+1}$$

$R=\frac{1}{z}$

$$\frac{.001k\left(\frac{1}{z}\right)^2}{(1+.001k)\left(\frac{1}{z}\right)^2-2\left(\frac{1}{z}\right)+1} \cdot \frac{z^2}{z^2}$$

$$\frac{.001k}{(1+.001k)\cdot 1-2z+1z^2}$$

a b c

$$\frac{2\pm\sqrt{4-4\cdot(1+.001k)\cdot 1}}{2(1+.001k)}$$

$$\frac{2\pm\sqrt{-.001k}}{2+.002k}$$

opps flipped these

(19)

$$\frac{2 \pm \sqrt{4 - 4 \cdot 1 \cdot (1 + .001k)}}{2 \cdot 1}$$

$$\frac{2 \pm \sqrt{0 - .004k}}{2}$$

$1 \pm \frac{\sqrt{.004}}{2}$ ↑ depends if k is ⊕/⊖

Cases

k ⊕ → complex

~~k ⊕~~

positive pole

keep oscillating

↑

unbounded

k ⊕ ~~and 2k~~ → ~~complex~~

↑

Magnitude at the pole $= \sqrt{\text{real}^2 + (\text{ipart})^2}$ = always ⊕ and > 1

always 1

$k = 0$ → bounded, monotonical → converges to some value

k ⊖ → $(>1, +)$ Unbounded monotonic

real life: turned away from wall faster

Special case - will never turn (think in real life)

(20)

is there a nice k value? -where converges monotonically

- there is no nice value of k where it converges

- no stable system

— no matter your k

(at this V, T)

HW Week 5

10/7

- doing this early - so lots of problems
- 1 question: System Behavior

Part 1 Stable?

- stable if bounded, trasient output converges to 0 as $n \to \infty$
- enter magnitude of dominate pole, if stable, if oscilatory
- ~~delete~~ note: alternating have osilation period of 2
- hint: use System Function class w/ floats
- (good, I wanted to practice this)

$$y[n] = \frac{5}{6} y[n-1] + y[n-2] + x[n]$$

$$Y = \frac{5}{6} YR + YR^2 + X$$

$$Y - \frac{5}{6} YR - YR^2 = X$$

$$Y\left(1 - \frac{5}{6}R - R^2\right) = X$$

$$\frac{Y}{X} = \frac{1}{1 - \frac{5}{6}R - R^2}$$

sf.SystemFunction(poly.polynomial[1], ... [1, -5.0/6, -1])

-1.5

(2)

Now -1.5 is

real + negitive → alternate signs

~~[illegible] → bounded / stable~~

~~magnitude~~

< 1 → don't care

$|> 1|$ → unbanded / unstable ✓

$|abs| \to 1.5$ ✓

oscilatory?

- what is this as defined above?
- say yes? ✗

no ✓

"; only if on real?

$$\frac{1}{1 + \frac{5}{4}R + \frac{3}{8}R^2}$$

$= -2$

$abs \to 2$

~~real + neg → alter~~

Same as above 2 ✗ ∈ ? even 2 was wrong?

no ✗

no ✗

(3)

Opps → wrong order

$[3.8/8, 5.0/4, 1]$

and I got 1st one wrong

- was 1.5

so real + positive → increase/decrease monotonically

- so is this ossilate no

back to #2

$-.75$

$.75$

must → yes

be → yes

why?

|abs| < 1 → bounded/stable

negitive → alternate signs

↑ ossilate yes?

$$y[n] = -\frac{3}{2}y[n-1] - \frac{9}{8}[n-2] + x[n]$$

$$Y = -\frac{3}{2}YR - \frac{9}{8}R^2Y + X$$

$$Y\left(1 + \frac{3}{2}R + \frac{9}{8}R^2\right) = X$$

$$\frac{Y}{X} = \frac{1}{1 + \frac{9}{8}R^2 + \frac{3}{2}R + 1}$$

(4)

$(-.75 + .75j)$

|abs| → .75

< 1 → bounded / stable

Complex → periodic

.75	$.75 + .75j$	1.06
⊗	⊗	✓
yes ⊗	no ✓ ← does	oscillation override
no ⊗	yes ✓	– no 2 is stable + oscillation and it is negitive

wolfram alpha

I get it now 1.06 is > 1 unbounded

$$\frac{1}{\frac{1}{2}R^2 + R + 1}$$

$(-.5 + .5j)$

.5	$.5 + .5J$	.707
⊗	⊗	✓
Yes ✓		
no ⊗	yes ✓ ← j negitive	

– so complex is seperate or duch periodic is oscilating
– stupid

Part 2 Diff Eq behavior

- choose which you think matches pic
- do I need to test them all!

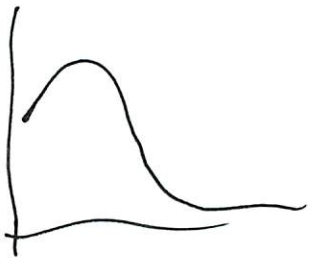
(5) Oh 4 ans → 4 graphs

A $y_{n+1} = \frac{13}{8} R - \frac{42}{64} y R^2 + x$

$\frac{y}{x} = \frac{1}{+\frac{42}{64} + 1}$ ← $\frac{13}{8}$ A

$= .875$
- bounded
- monotonic

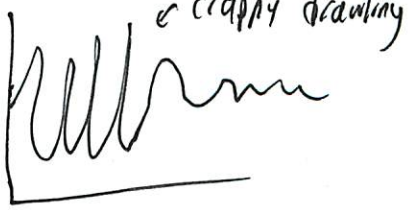
4 ✓



#B $-.875$
- bounded
- ossiclate

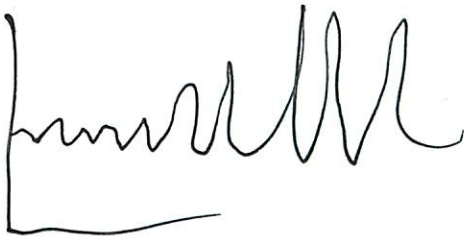
set up by following pattern

2 ✓



#C -1.125
unbanded
Ossiate

1 ✓



#D 1.125
unbandied
monotonic

3 ✓