How To Do Linear Processing in a Nonlinear World

Have Your Cake and Eat it Too

Sam Roweis, Caltech

Two Potential Curses

- Consider the task of doing some linear filtering on an input signal $u(t)$ to give an output signal $y(t)$.
- The problem is easy to do if you have the input signal $u(t)$ directly available AND the primitives:
  1. Add/Subtract two signals
  2. Multiply a signal by a constant
  3. Memory Element (eg. integrator/differentiator)
- Imagine that the evil wizard EXPONEN has taken away your linear gain element and replaced it with the primitive EXP().
  Q: What can you do to recover from this injustice and still do linear processing on $u(t)$?
  A: It turns out that taking the LOG of the inputs will save you.
- Alternately, imagine that the evil wizard SENSOR has taken the log of your inputs.
  A: It turns out that replacing your linear gain element with the primitive EXP() wil save you.

When Can We Build Linear Systems ?

- Consider the task of doing some linear filtering on an input signal $u(t)$ to give an output signal $y(t)$.
- The problem is easy to do if you have the input signal $u(t)$ directly available AND the primitives:
  1. Add/Subtract two signals
  2. Multiply a signal by a constant
  3. Memory Element (eg. integrator/differentiator)
- The problem is much harder but often still possible to do if either:
  1. You are forced to work in a design space that does not have the linear gain primitive available but instead has a (possibly nonlinear) function $E[\cdot]$ instead (e.g. silicon).
  2. You do not have the input $u(t)$ availalbe directly but instead only have $L[u(t)]$ where $L[\cdot]$ is some input transformation beyond your control.
- *Log Domain Filtering* [R.Adams '79, D.Frey '94] solves these two problems when:

  $E[\cdot] = exp(\cdot)$ or $L[\cdot] = log(\cdot)$

Why Log, Exp, and LTI ?

- Goal: design a system that has *linear* input output behaviour using *nonlinear* primitives.
- Consider specifically the design of stable LTI (*linear, time-invariant*) systems:

$$H(s) = \frac{q(s)}{p(s)} = \frac{q_m s^m + \ldots + q_0}{p_n s^n + \ldots + 1}$$

(for polynomials p and q with $deg(p) \geq deg(q)$, with no poles (roots of p)in Re$(s) \geq 0$)

- Importance ?
Any system governed by a linear o.d.e:

$$p_n y^{(n)}(t) + \ldots + y(t) = q_m u^{(m)}(t) + \ldots + q_0 u(t)$$

$$p\left(\frac{d}{dt}\right) y(t) = q\left(\frac{d}{dt}\right) u(t)$$

is a LTI system and has a corresponding TF $H(s)$

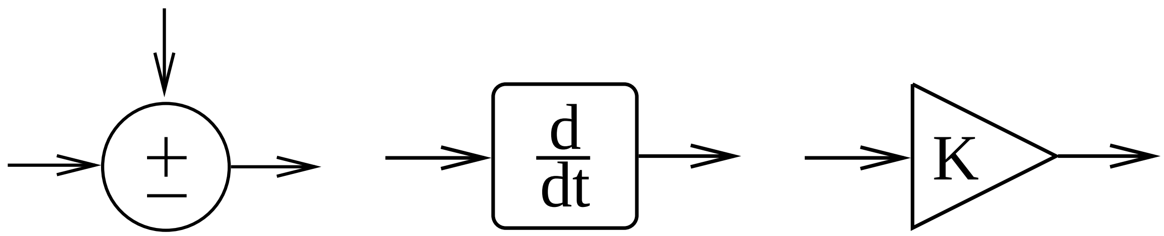
- Also superposition and input/output scaling:

$$h\left(A \cdot u_1(t) + B \cdot u_2(t)\right) = A \cdot h\left(u_1(t)\right) + B \cdot h\left(u_2(t)\right)$$

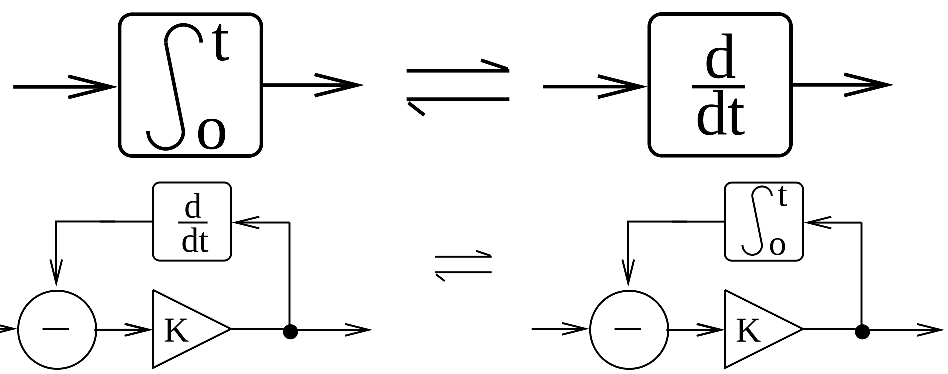
- Log and Exp occur often as input sensor or computing primitive constraints.

Conventional Primitives

- Consider the computational primitives:



- Having a differentiator gives you an integrator and vice versa. Only one needs to be a primitive:



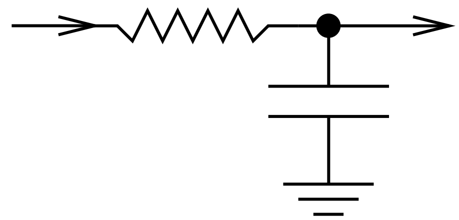
$$K[u(t) - \dot{y}(t)] = y(t) \qquad K[u(t) - \textstyle\int_0^t y(t)] = y(t)$$

$$\frac{Y(s)}{U(s)} = \frac{1}{s + 1/K} \approx \frac{1}{s} \qquad \frac{Y(s)}{U(s)} = \frac{1}{1/s + 1/K} \approx s$$

- Notice: integrator/differentiator blocks require memory. The other blocks are memoryless.
- The integrator remembers its running integral, the differentiator remembers a past value to compare with.

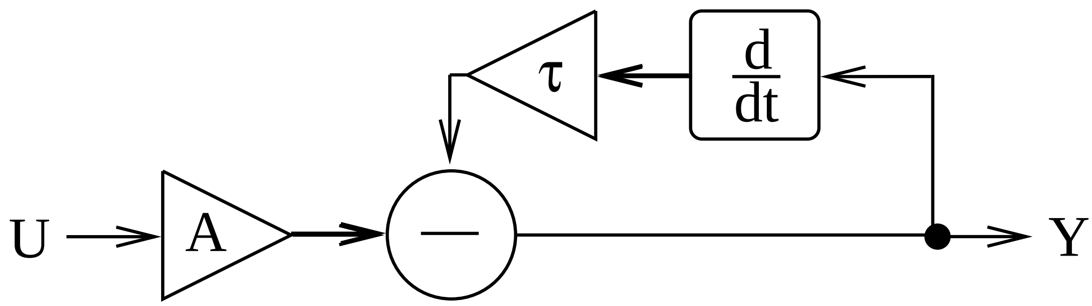
A Simple Example

- Say we want to build a low pass filter:

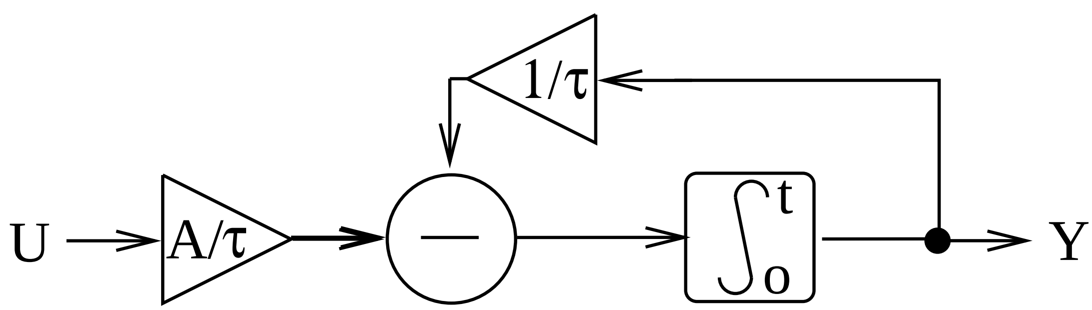


$$\tau \dot{y} + y = au$$
$$H(s) = \frac{A}{\tau s+1}$$

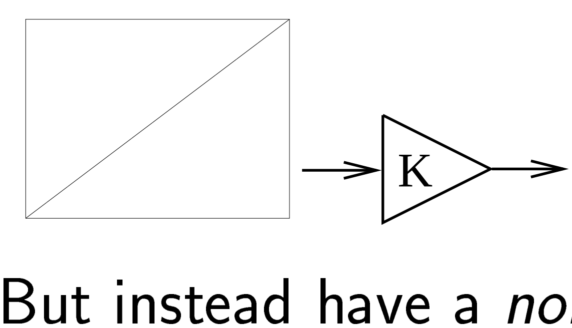
- One way to build it:

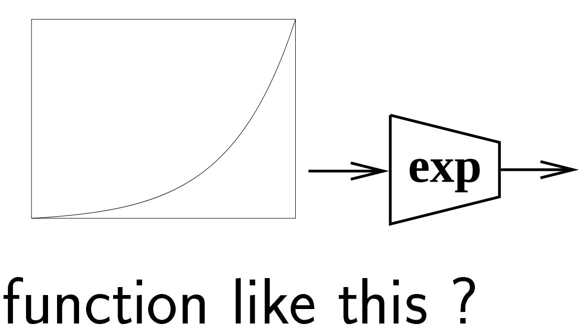


- An equivalent way:



- What happens if we do not have a nice linear gain element like this:

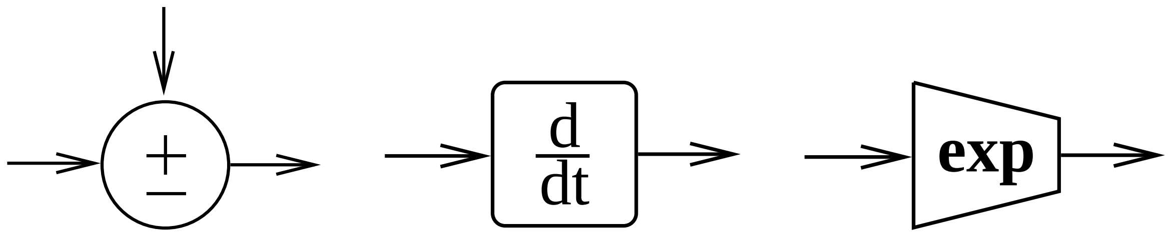




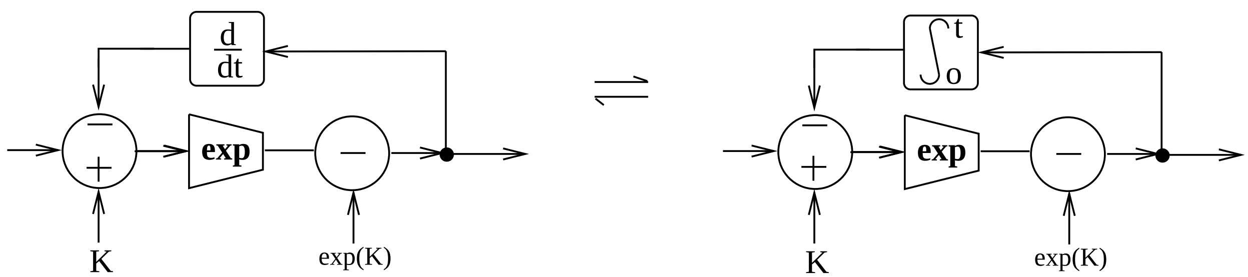
But instead have a *nonlinear* function like this ?

Nonlinear Primitives

- What can we say about these primitives ?

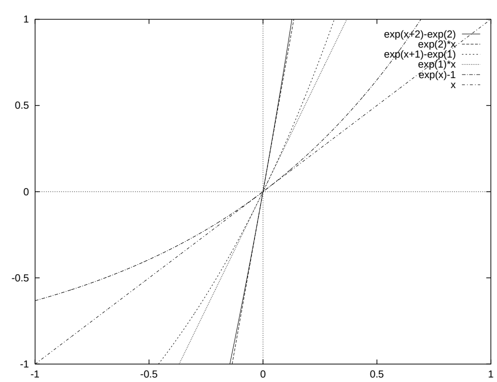


- Notice: It is still true that having a differentiator gives you an integrator and vice versa.



This is because all we need to do is sense the sign of the error. For big values of K,

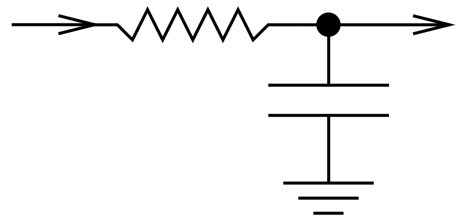
$$e^{x-K} - e^{K} \approx e^{K}x \approx e^{K}sign(x)$$



Warning: this does not mean we can replicate the gain element Kx. Only in the limit as $K \to \infty$ can we use this trick.

Lowpass Filter Revisited

- Let us redo the low pass filter with $exp()$:



$$\tau\dot{y} + y = au$$
$$H(s) = \frac{A}{\tau s+1}$$

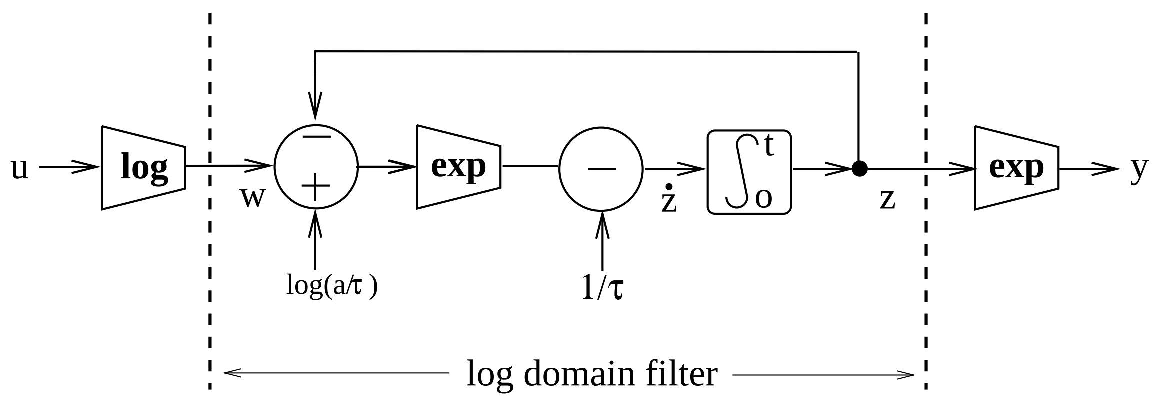
- Define new variables z and w according to

$$y = e^z,\ u = e^w \quad \Rightarrow \quad z = \log(y),\ w = \log(u)$$

And now the differential equation becomes:

$$\dot{z} = e^{w-z+\log(a/\tau)} - \frac{1}{\tau}$$

- This motivates the *log domain* lowpass filter:



- Notice:

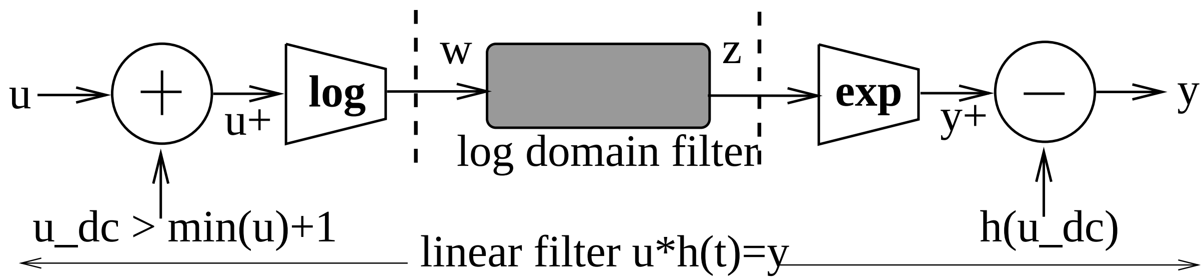
1. We need to get the log of the input somehow.
2. The input u and output y have now been constrained to be always positive.

Log Domain Filtering

- In log domain filtering, we find a mapping from the *log* of the input to the *log* of the output such that the end-to-end mapping from input to output is linear system of our choice.

  Input:u Log Input:w Log Output:z Output:y

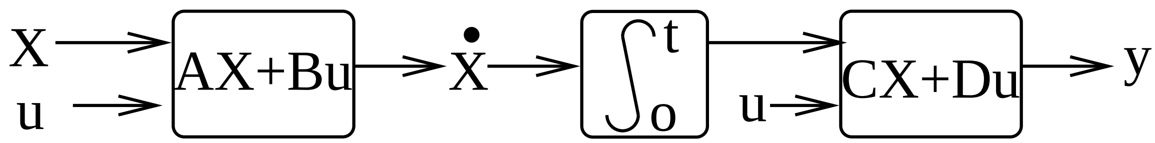
- This seems to constrain inputs and outputs to be positive, but we can always remap them:



- Given an eponential nonlinearity but no linear gain element it is still possible to synthesize *any* stable LTI system exactly.
- Caveats:
  1. The bounds on the inputs must be known.
  2. We need to “cheat” and allow one log operation per input dimension as part of the preprocessing, but the log domain filter only uses exponentials.

General LTI System Synthesis

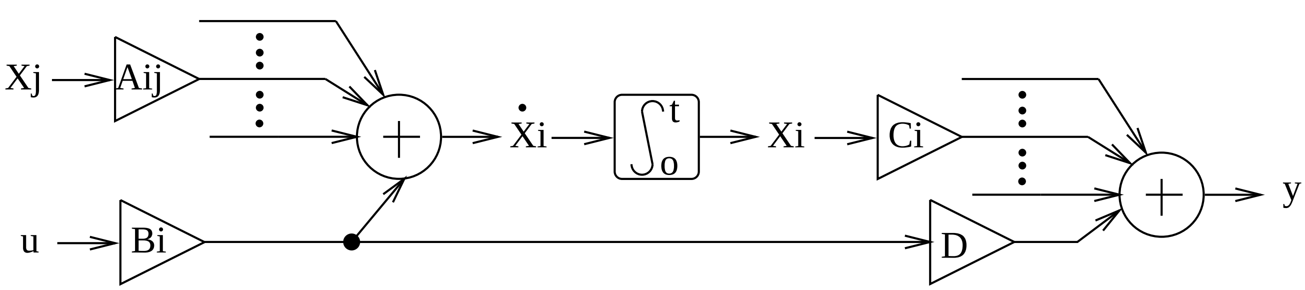
- Make use of *state space* realization.
- A *state vector* X is any set of quantities such that knowing X at t_0 and the input u for $t \geq t_0$ allows us to compute the output y for $t \geq t_0$.
- Imagine two cascaded systems, one with memory which evolves the state vector given the input, and one memoryless which computes the output from the state:



$\dot{X} = AX + Bu$ A is n by n, B is n by 1

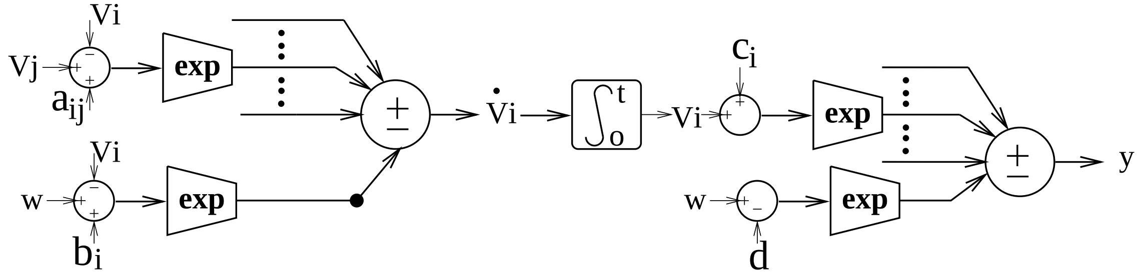
$y = CX + Du$ C is 1 by n, D is scalar

- Knowing A, B, C, D is the same as knowing $h(t)$ or $H(s)$ or the governing differential equation.
- With linear primitives, we can implement the state model directly:



Log Domain State Space Models

- Question: Was the lowpass filter a fluke, or can we synthesize more complicated linear systems using nonlinear elements ?
- Surprising result: we make *all* causal, stable LTI systems using *log state* variables.
- Define a new state vector V such that $x_i = e^{v_i}$
  The new state dynamics are given by:
  $$\dot{v}_i = \sum_{j=1}^{n} (\pm e^{v_j - v_i + a_{ij}}) \pm e^{w - v_i + b_i}$$
  And the new output function is:
  $$y = \sum_{i=1}^{n} (\pm e^{v_i + c_i}) \pm e^{w-d}$$
  where $a_{ij} = \log|A_{ij}|, b_i = \log|B_i|$,
  $c_i = \log|C_i|, d = \log|D|$ and the choice of signs $\pm$ follow the signs of the original quantities
- New model now can be implemented directly:



A Serious Problem

- The log domain state model constrains the original state variables x_i to be strictly positive.
- Solution: define a new state vector $\tilde{X} = MX$ for an invertible linear map M. New dynamics:

$$\dot{\tilde{X}} = (MAM^{-1})\tilde{X} + (MB)u$$
$$y = (CM^{-1})\tilde{X} + Du$$

- Pick M so that the equilibruim DC state (offset input only, $\dot{X} = 0$) is all positive:

$$0 = (MAM^{-1})\tilde{X}_{dc} + (MB)u_{dc}$$
$$\Rightarrow \tilde{X}_{dc} = -(MA^{-1}B)u_{dc}$$

- Limits on the new state variable excursions can now established [eg. Vidyasgar for derivation]:

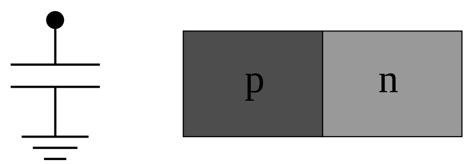
$$\max_{\forall t} |\tilde{X}_i - \tilde{X}_{i_{dc}}| \leq \max_{\forall s} |H_i(s)| \cdot \max_{\forall t} |u|$$

where $H_i(s)$ is the i^{th} row of $(sI - A)^{-1}B$

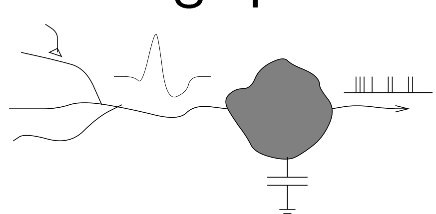
- Simply increase u_{dc} until $\tilde{X}_i$ are always positive.

Some Physical Realizations

- Electronics (almost exact):
  The current across a pn junction is exponential in the potential difference. A capacitor integrates charge (or differentiates voltage). Currents sum or difference at circuit nodes.

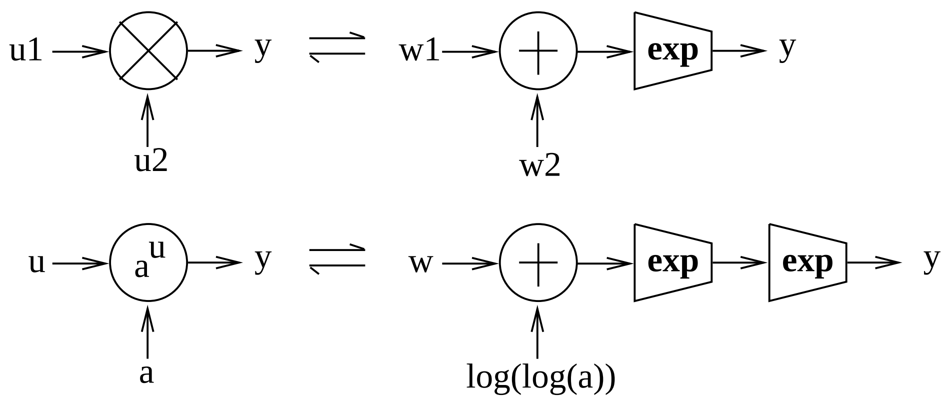


- Biology (extremely tenuous):
  Cell firing rate proportional to membrane voltage
  Post synaptic potentials summate in dendrites.
  Soma membrane capacitance integrates EPSPs or IPSPs from incoming spikes.



Beyond Linear Systems

- With log domain filtering we can eat our cake and have it whole too.
- We can implement many nonlinearities using the exponential primitive as well as all linear systems.
- Examples: multiplication, raise to a power, etc.



- The function $f = \dot{\theta} e^{-\alpha\theta}$