Difference Equations and Convolution

Background

A *differential equation* is a function which

- Exists for all time (i.e. is an analog signal), and
- Is of the form

$$\frac{d^3y}{dt^3}+a_2\frac{d^2y}{dt^2}+a_1\frac{dy}{dt}+a_0=b_3\frac{d^3x}{dt^3}+b_2\frac{d^2x}{dt^2}+b_1\frac{dx}{dt}+b_0x$$

A *difference equation* is a function which

- Exists only at discrete times (k = 0, 1, 2, ...), and
- Is of the form

$$y_{k+3}+a_2y_{k+2}+a_1y_{k+1}+a_0y_k=b_3x_{k+3}+b_2x_{k+2}+b_1x_{k+1}+b_0x_k$$

Examples of difference equations are:

Car Loan:

- At month 0 (k=0),. borrow $10,000
- Each month, the loan gains 1% (12% interest per year - really bad loan)
- Each month, you pay $100.

The amount you owe each month is

$$y_k=1.01y_{k-1}-100$$

Software Program:

- Every 10ms, you measure temperature (xk)
- To reduce noise, average the last 3 measurements

$$y_k=\tfrac{1}{3}(x_k+x_{k-1}+x_{k-2})$$

Markov Chain:

Suppose you're playing a tennis match. The probability of winning any given game is 60%. To with the match, you have to be up by 2 games.

If you let the states be

$$X = \begin{bmatrix} \text{up 2 games} \\ \text{up 1 game} \\ \text{tied} \\ \text{down 1 game} \\ \text{down 2 games} \end{bmatrix}$$

Then the probability of being at state X at time k is

$$X_k = \begin{bmatrix} 1 & 0.6 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & & \\ & 0.4 & & 0.6 & \\ & & 0.4 & & 0 \\ & & & 0.4 & 1 \end{bmatrix} X_{k-1}$$

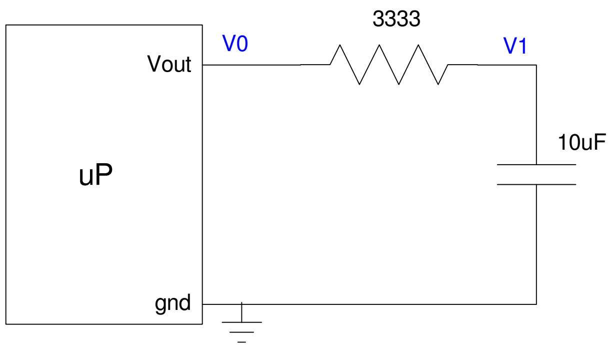
For example, if you were initially tied, there is a 60% change you'll be +1 after one game, 40% change you'll be -1 after one game

$$X_k = \begin{bmatrix} 1 & 0.6 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0 \\ 0 & 0.4 & 0 & 0.6 & 0 \\ 0 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.6 \\ 0 \\ 0.4 \\ 0 \end{bmatrix}$$

Our present tools don't work well for such systems. We need a new tool.

Microprovcesor Control

Suppose a microprocessor is driving a dynamic system. For the sake of simplicity, assume the microprocessor updates its output every 100ms and the dynamic system is an RC filter:



By voltage division, the voltage at V1 will be

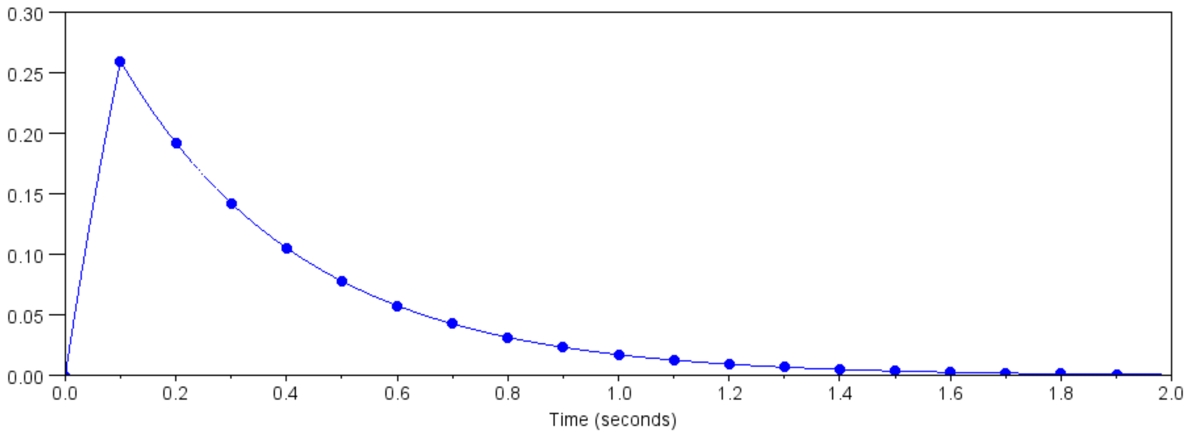
$$V_1(s) = \left(\frac{1/Cs}{R+1/Cs}\right) V_0(s)$$

$$V_1 = \left(\frac{3}{s+3}\right) V_0$$

If the mircoprocessor outputs 1V for 100ms (one clock for the microprocessor), then for the first 100ms, V1 will be

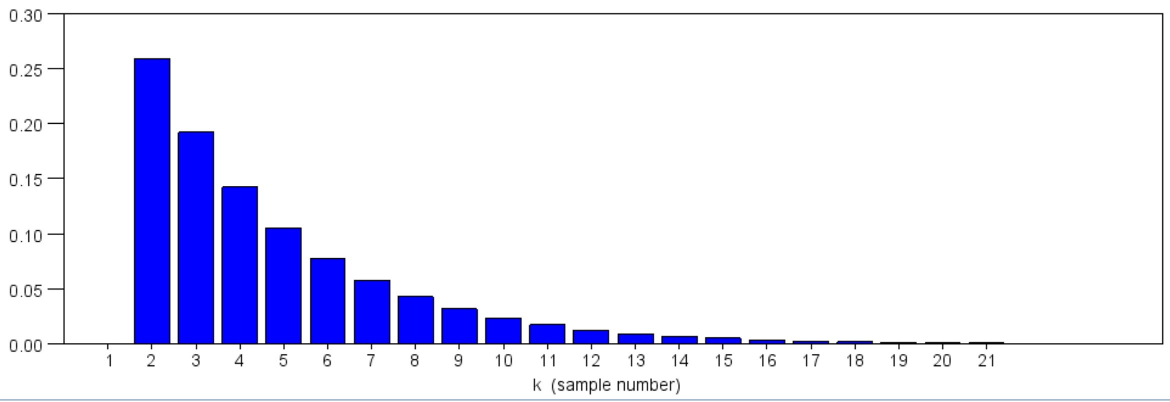
$$V_1 = \left(\frac{3}{s+3}\right)\left(\frac{1-e^{-0.1s}}{s}\right)$$

$$v_1(t) = (1-e^{-3t})u(t) - (1-e^{-3(t-0.1)})u(t-0.1)$$



v1(t) for a 100ms pulse from the microprocessor.

Note that, relative to the microprocessor, this is a discrete-time system: all the microprocessor sees is the voltage at the sampling times (shown by the blue dots). Relative to the microprocessor, v1(t) looks like:



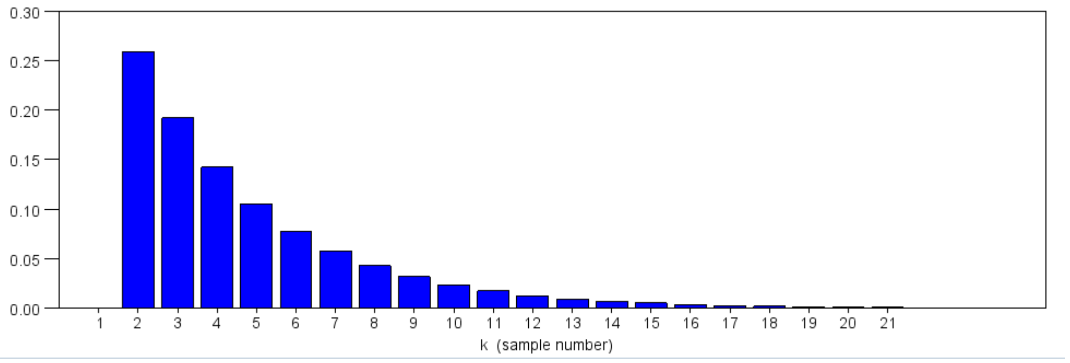
V1 at time k, where k is the sample number (t = k*T)

Discrete-Time Convolution

For discrete-time systems, the delta function is defined as

$$\delta(k) = \begin{cases} 1 & k = 0 \\ 0 & otherwise \end{cases}$$

This corresponds to the microprocesor outputting 1V for one sample (100ms) in the previous example. The *impulse response* is the response of a dynamic system to an impulse function. The previous bar graph is likewise the impulse response of the RC filter.



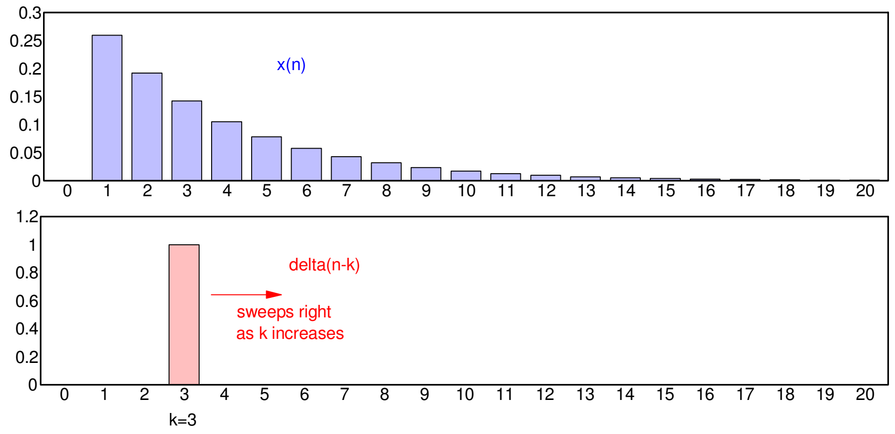
Impulse Response, h(k)

Suppose the input was something other than an impulse. Like we did with LaPlace transforms, you can determine x(k) in terms of the delta function as

$$x(k) = \sum_n x(n)\delta(k-n)$$

Explanation:

- x(n) is the function we're trying to represent.
- $\delta(k-n)$ is a delta function, delayed by n samples.
- When n = 0, you pick off x(0)
- As n increases, the delta function sweeps right, picking off each value of x(k)



This is also known as *discrete time convolution:*

$$x(k) = x(k) ** \delta(k)$$

$$x(k) = \sum_n x(n)\delta(k-n)$$

A function convolved with a delta function returns that function.

The function whose impulse response is $\delta(k)$ is one (i.e. a wire)

$$\delta(k) = 1 \cdot \delta(k)$$

This means that if you short the input to the output, the output will equal the input (duh).

Suppose you connect the input to the output with something whose impulse response is h(k), as in the RC filter from before. Then, the output will be

$$y(k) = \sum_n x(n)h(k-n)$$

It works a little better to swap the order:

$$y(k) = \sum_n h(n)x(k-n)$$

For example, suppose the output of the microprocessor was u(k)

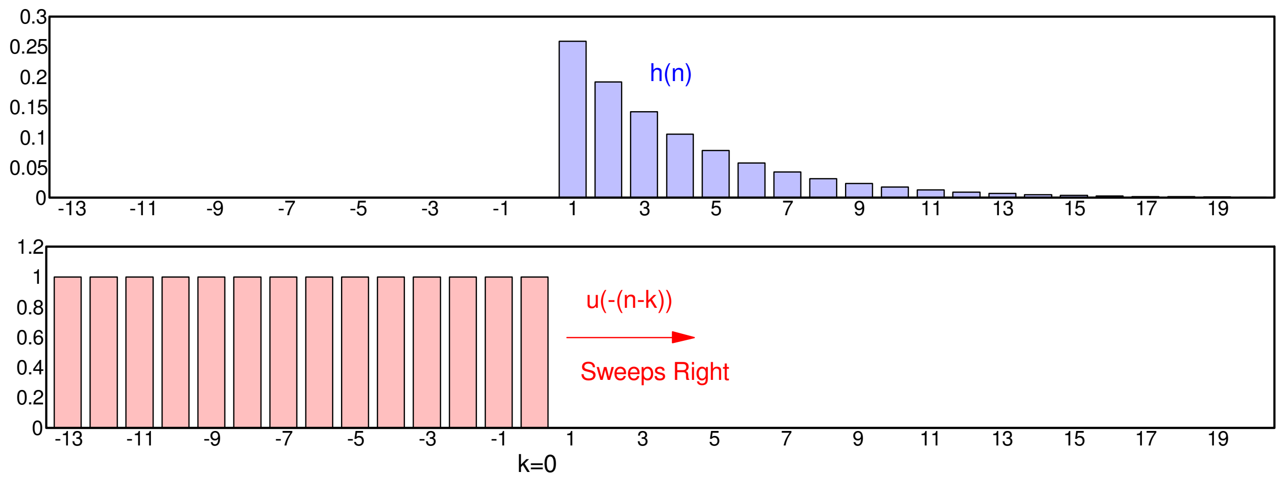
$$x(k) = u(k) = \begin{cases} 1 & k \ge 0 \\ 0 & otherwise \end{cases}$$

Then

$$y(k) = \sum_n h(n)u(k-n)$$

Graphically, this is

- The impulse function, h(n)
- Multiplied by u(n), flipped (minus n), delayed by k



y(k) is the sum of h(n), multiplied by u(-(n-k)) a.k.a. h(k) ** u(k)

Spread-sheets work really well for this type of problem:

- Input the impulse response in column #3
- Column #4 is column #3 delayed by one
- Column #5 is column #4 delayed by one, etc
- y(k) is the sum of column #3 to infinity

k	y(k)	h(k)	k(k-1)	h(k-2)	h(k-3)	h(k-4)	h(k-5)
0	0	0	0	0	0	0	0
1	0.26	0.26	0	0	0	0	0
2	0.44	0.18	0.26	0	0	0	0
3	0.56	0.13	0.18	0.26	0	0	0
4	0.65	0.09	0.13	0.18	0.26	0	0
5	0.71	0.06	0.09	0.13	0.18	0.26	0
6	0.75	0.04	0.06	0.09	0.13	0.18	0.26
7	0.78	0.03	0.04	0.06	0.09	0.13	0.18
8	0.8	0.02	0.03	0.04	0.06	0.09	0.13
9	0.82	0.02	0.02	0.03	0.04	0.06	0.09
10	0.83	0.01	0.02	0.02	0.03	0.04	0.06
11	0.84	0.01	0.01	0.02	0.02	0.03	0.04
12	0.84	0.01	0.01	0.01	0.02	0.02	0.03
13	0.84	0	0.01	0.01	0.01	0.02	0.02
etc.	:	:	:	:	:	:	:

Doing this in Matlab or by hand is really a pain. We need a better tool.

For continuous time systems, that tool was LaPlace transforms.

- Continuous-time convolution becomes multiplication in the LaPlace domain.

For discrete-time systems that tool is z-transforms.

- Discrete-time convolution becomes multiplication in the z-domain.

Convolution & Multiplying Polynomials

One sidelight - multiplying polynomials is actually convolution. For example, find the product of

$$(3+4x+5x^2)(6+7x+8x^2) = ?$$

This is the convolution of

```
[3 4 5] ** [6 7 8]
```

$$\begin{array}{ccccc} & & 6\ + & 7x\ + & 8x^2 \\ 5x^2\ + & 4x\ + & 3 & & \end{array} \quad = 18$$

$$\begin{array}{cccc} & 6\ + & 7x\ + & 8x^2 \\ 5x^2\ + & 4x\ + & 3 & \end{array} \quad = 24x + 21x = 45x$$

$$\begin{array}{ccc} 6\ + & 7x\ + & 8x^2 \\ 5x^2\ + & 4x\ + & 3 \end{array} \quad = 30x^2 + 28x^2 + 24x^2 = 82x^2$$

$$\begin{array}{cccc} 6\ + & 7x\ + & 8x^2 & \\ & 5x^2\ + & 4x\ + & 3 \end{array} \quad = 35x^3 + 32x^2 = 67x^3$$

$$\begin{array}{ccccc} 6\ + & 7x\ + & 8x^2 & & \\ & & 5x^2\ + & 4x\ + & 3 \end{array} \quad = 40x^4$$

answer

$$\boxed{(3+4x+5x^2)(6+7x+8x^2) = 18 + 45x + 82x^2 + 67x^3 + 40x^4}$$

Convolution and Markov Chains:

Suppose you're playing a game. Each round, there is a

- 60% chance you'll win
- 40% chance you'll lose.

The game is 3 rounds. What is the probability you'll have 0, 1, 2, or 3 wins?

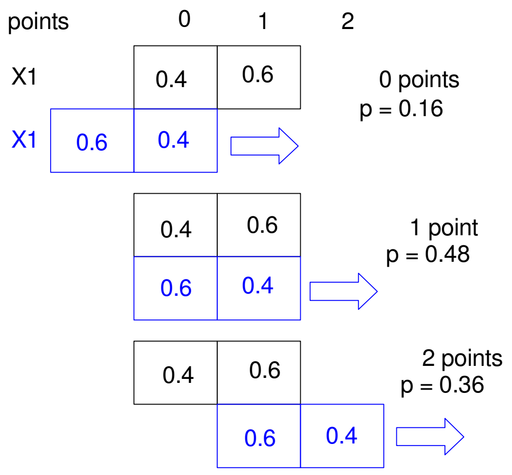
Solution: This is the same as multiplying polynomials.

Round #1 After one round, you have a 40% chance of 0 points, 60% chance of one point

points	0	1	2
X1	0.4	0.6	0

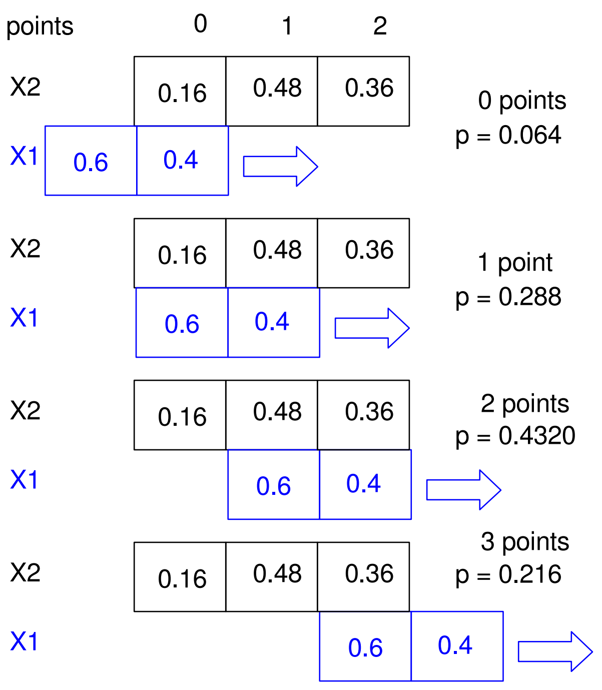
After round #1, you have a 60% chance of having 1 point, 40% chance of 0 points

Round #2: X2 = X1 ** X1



After round #2, the probability of [0, 1, 2] points is [0.16, 0.48, 0.36]

Round #3: X3 = X2 ** X1



After 3 rounds, the probability of [0, 1, 2, 3] points is [0.064, 0.288, 0.432, 0.216]

etc.