In4073
Embedded Real-Time Systems

Introduction to Digital Filtering

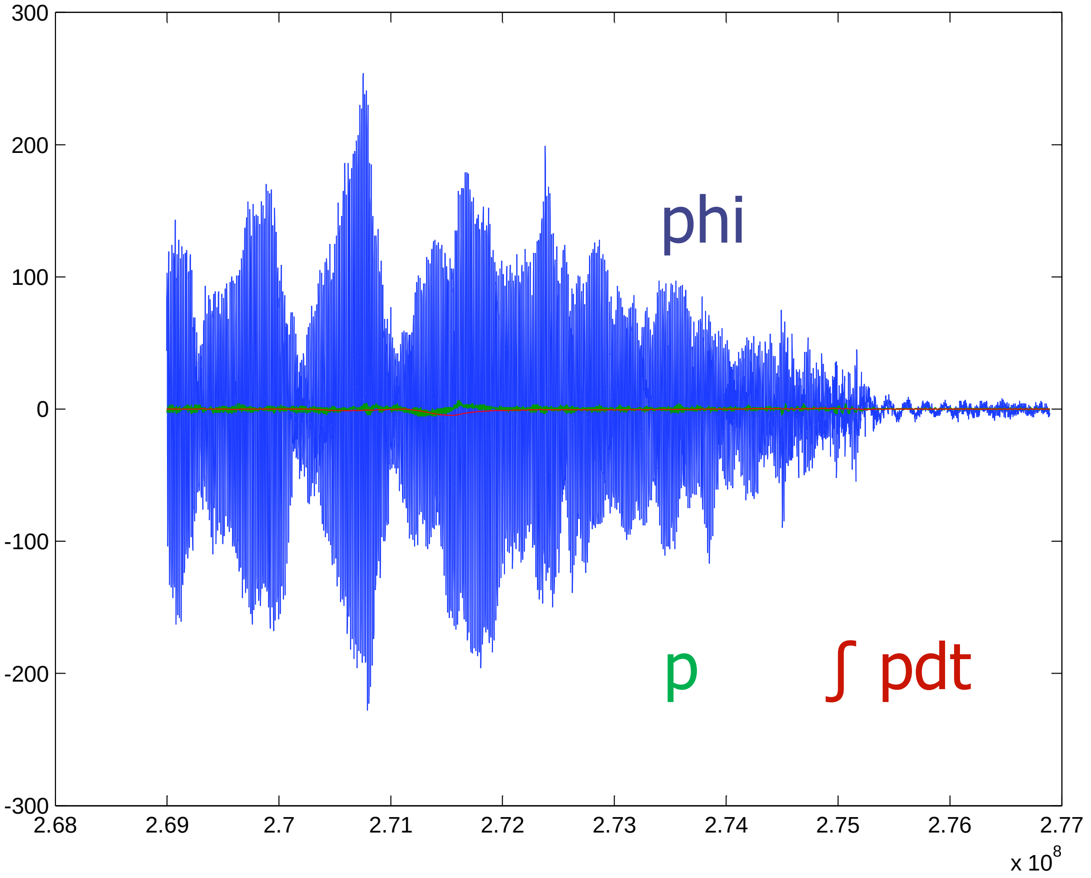
Outline

- Introduction
- Z Transform
- FIR Filters
- IIR Filters
- Fixed-point Implementation
- Kalman Filter

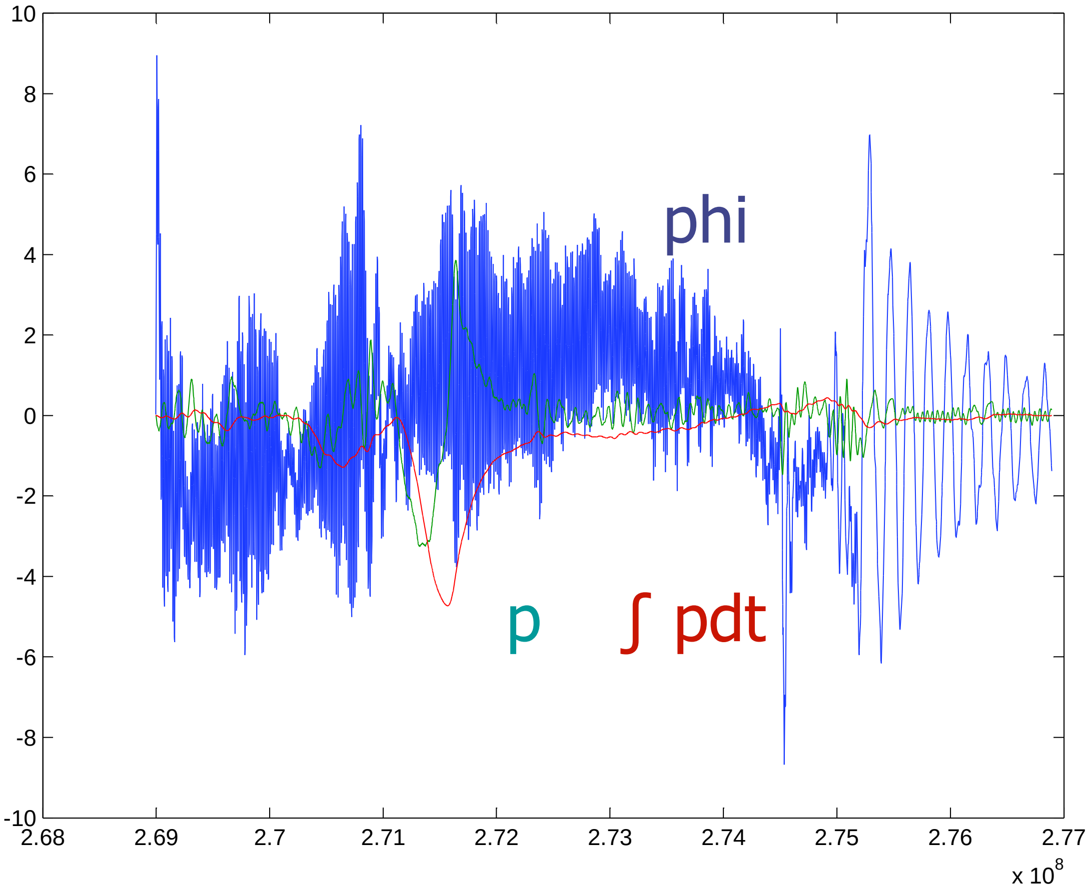
Why Signal Processing?

- Improve/restore media content
  - Compression/Decompression
  - Audio filtering (bass, treble, equalization)
  - Video filtering (enhancement, contours, ..)
  - Noise suppression (accel, gyro data)
  - Data fusion (mixing accel + gyro data)
- By digital means: DSP

Example: QR Sensor Signals phi, p



After some low-pass filtering



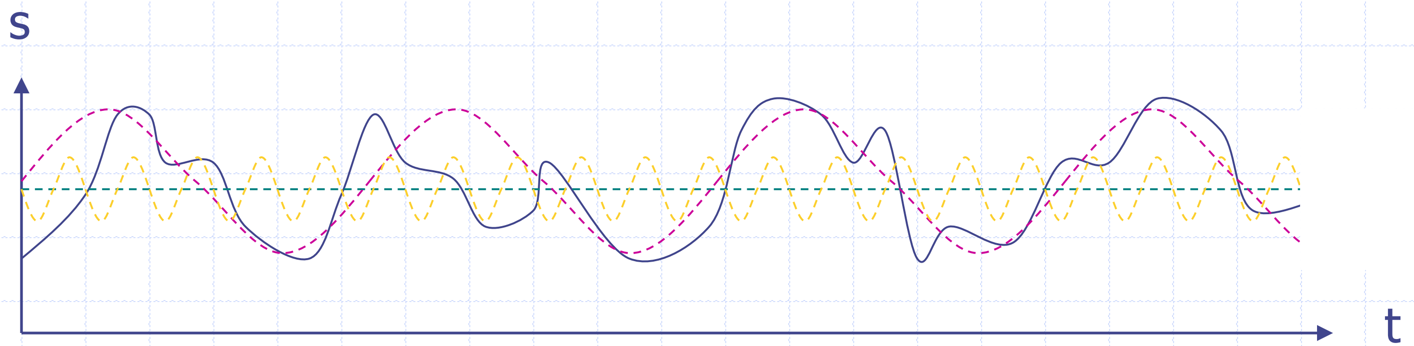
DSP is Everywhere

- Cell Phone
- TV
- Plant Control
- Climate Control
- Automotive
- Copiers, Wafer Scanners
- Model Quad Rotors ...

Objectives of this Crash Course

- Appreciate the benefits of Digital Filtering
- Understand *some* of the basic principles
- Communicate with DSP engineers
- Implement your own filters for the QR

Signals and Frequency Synthesis

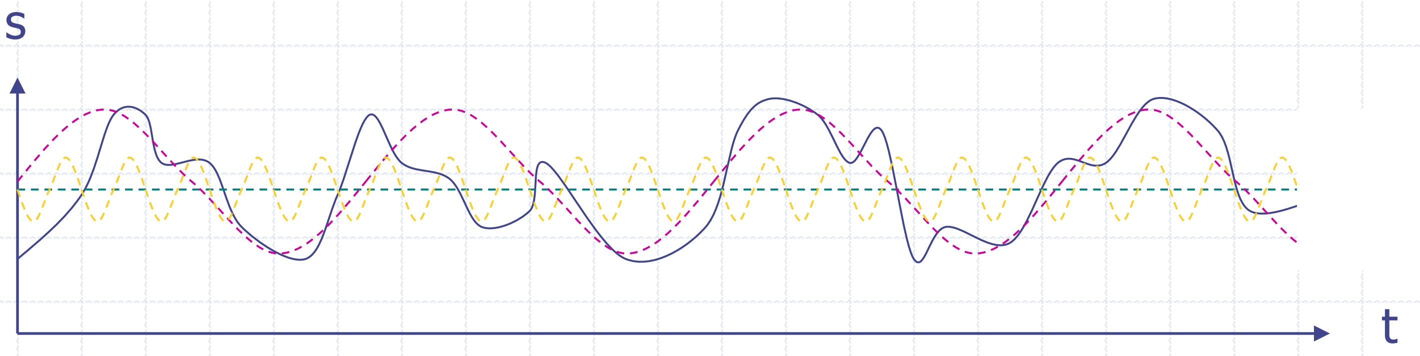


Usually signals (such as s) are composed of signals with many frequencies.
For instance, s contains

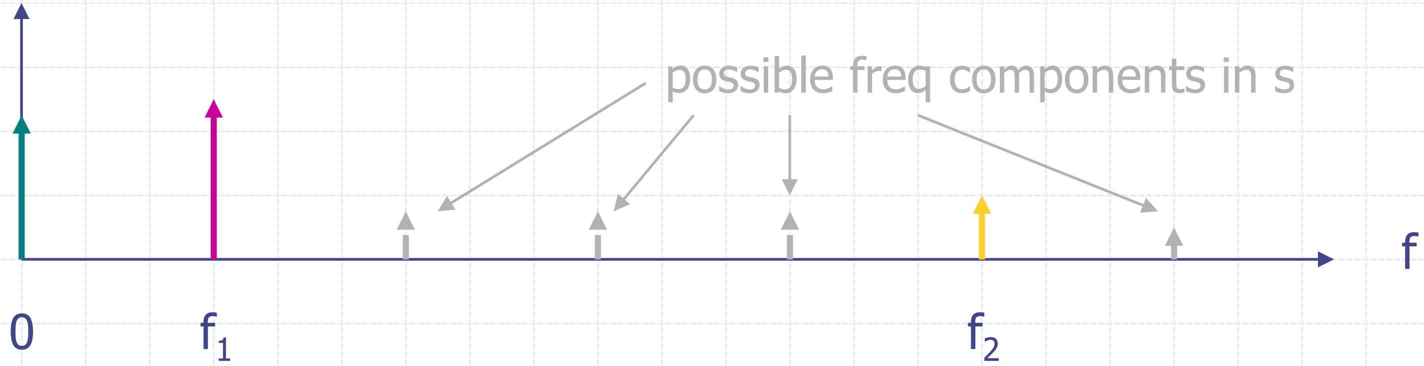
- 0 Hz component (green dashed line)
- lowest freq component (purple dashed line)
- higher freq component (yellow dashed line)
- and others

Fourier: Any *periodic* signal with base frequency f_b
can be constructed from sine waves with frequency f_b, $2f_b$, $3f_b$, …

Frequency Spectrum

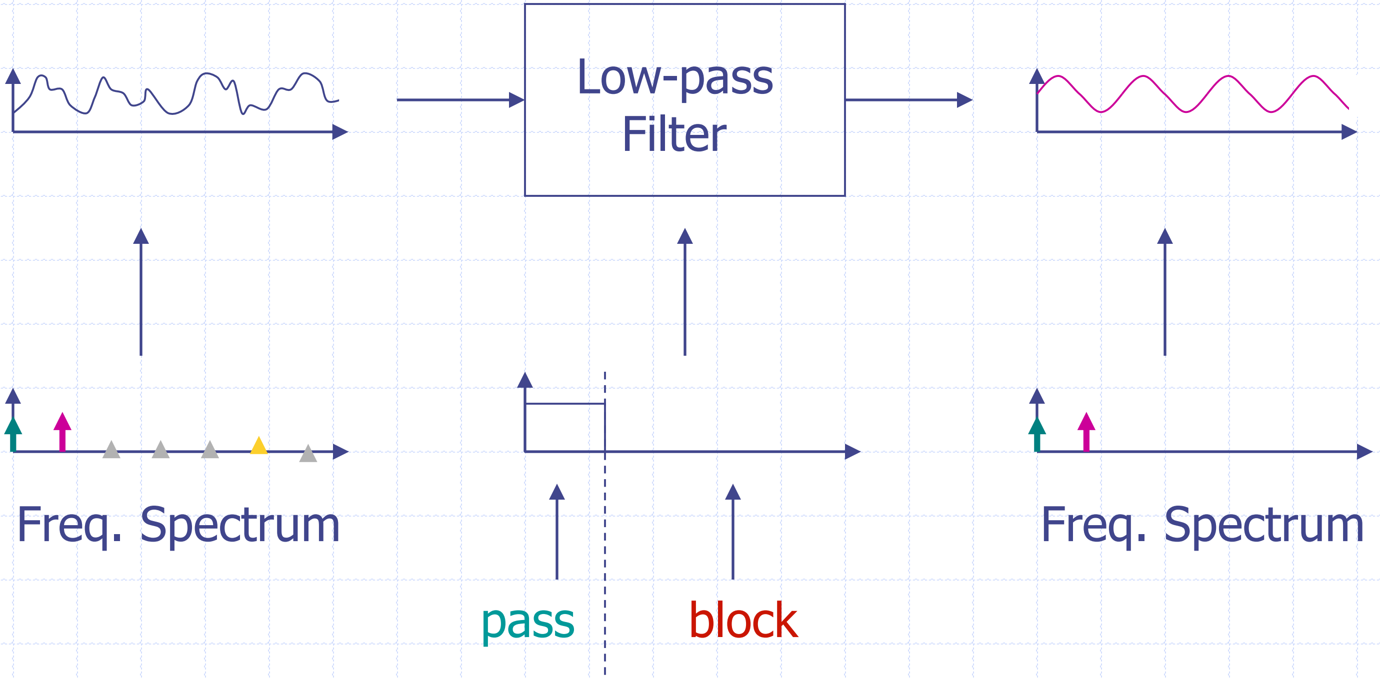


The frequency spectrum of s is:



Filter: Frequency Response

Often filters are designed to filter frequency components in a signal



Low-pass Filter

Freq. Spectrum

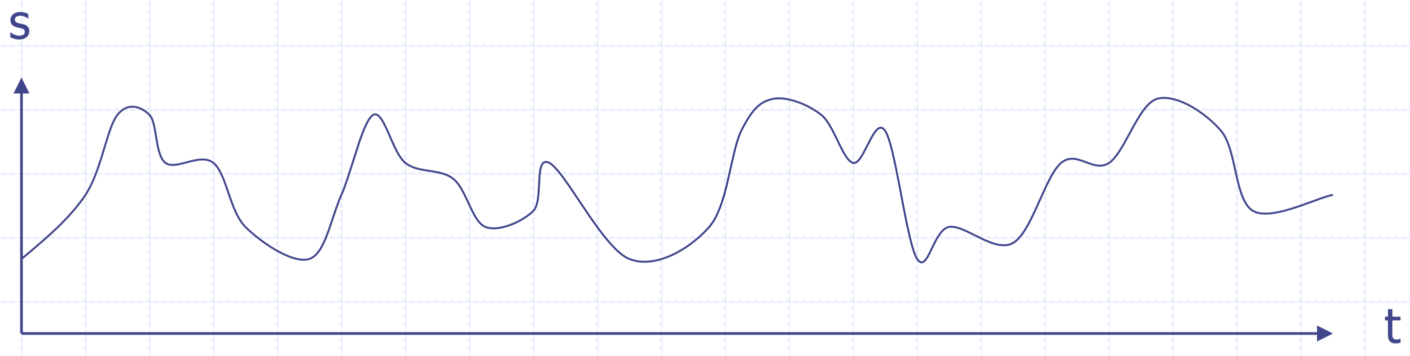
pass

block

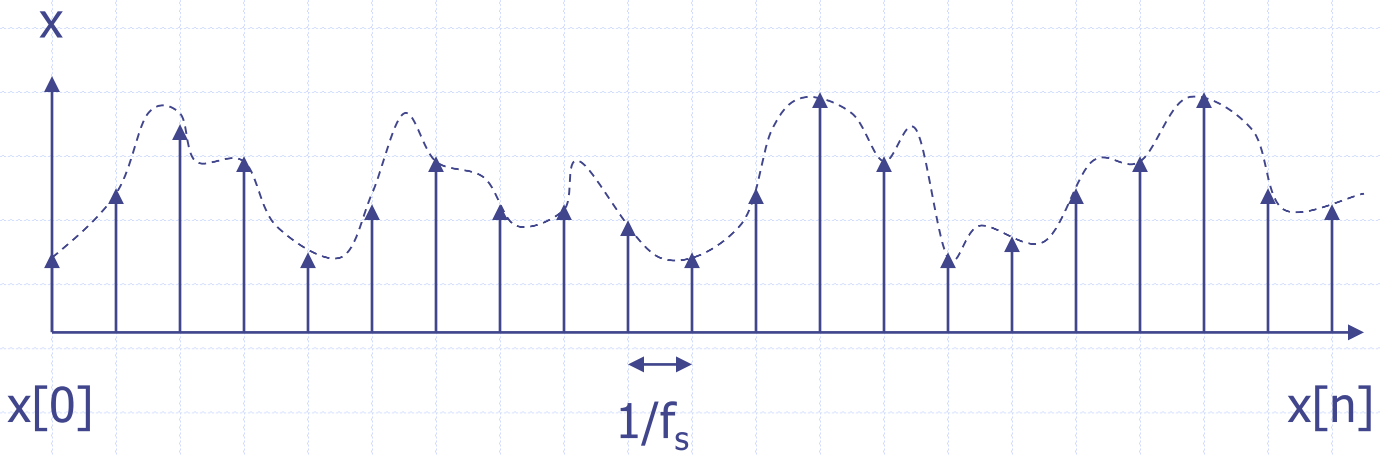
Freq. Spectrum

Filter's *Frequency Response*

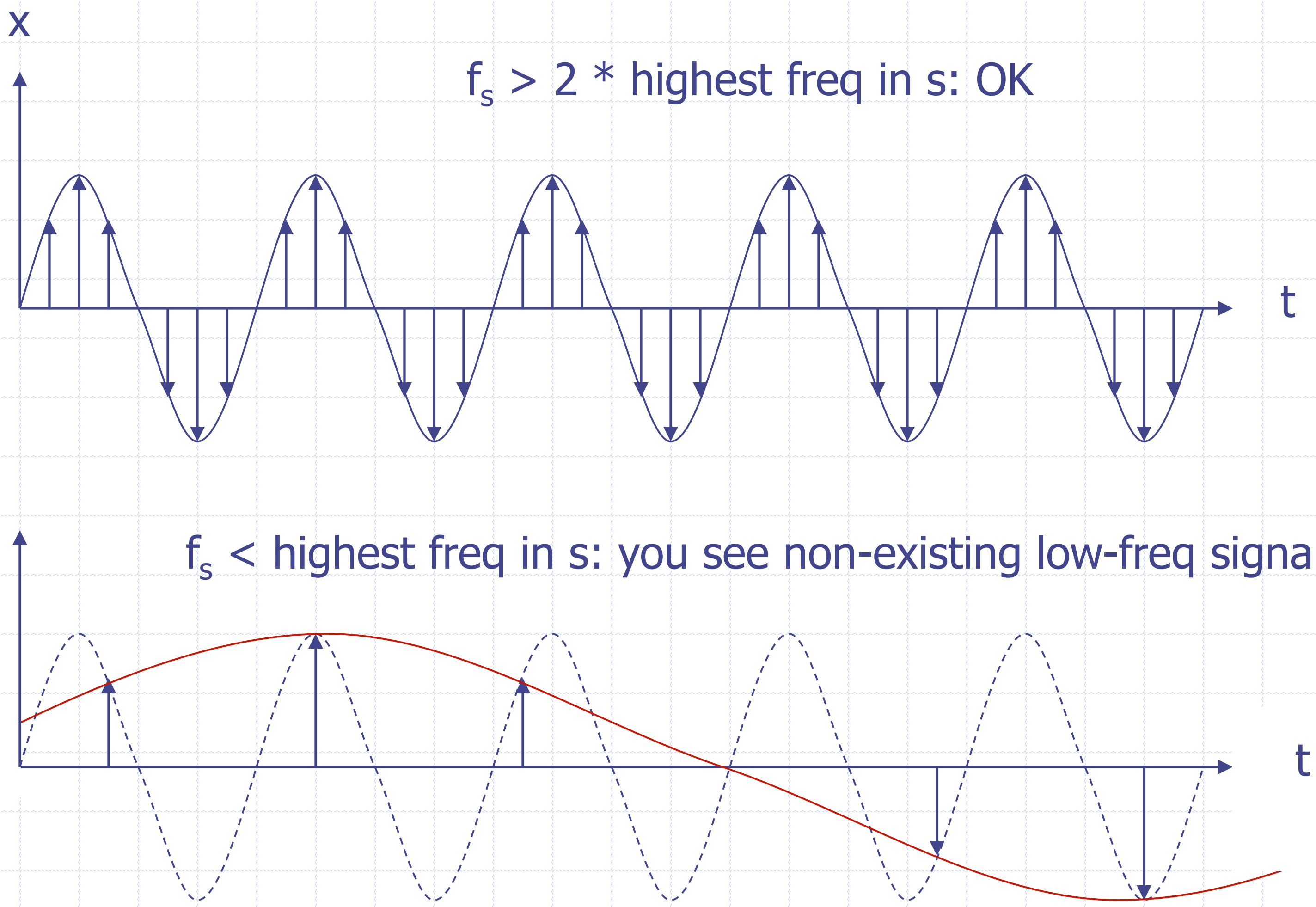
Sampling A Signal



s sampled at *discrete* time intervals (sample frequency f_s): x[n]



Sampling: Avoid Aliasing

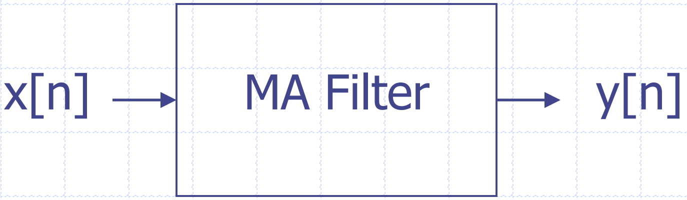


f_s > 2 * highest freq in s: OK

f_s < highest freq in s: you see non-existing low-freq signal(s)!

Example Filter: Moving Average

$y[n] = 1/3\ x[n] + 1/3\ x[n-1] + 1/3\ x[n-2]$



```
x[0] = get_sample();
y[0] = (x[0]+x[1]+x[2])/3;
put_sample(y[0]);
x[2] = x[1]; x[1] = x[0];
```

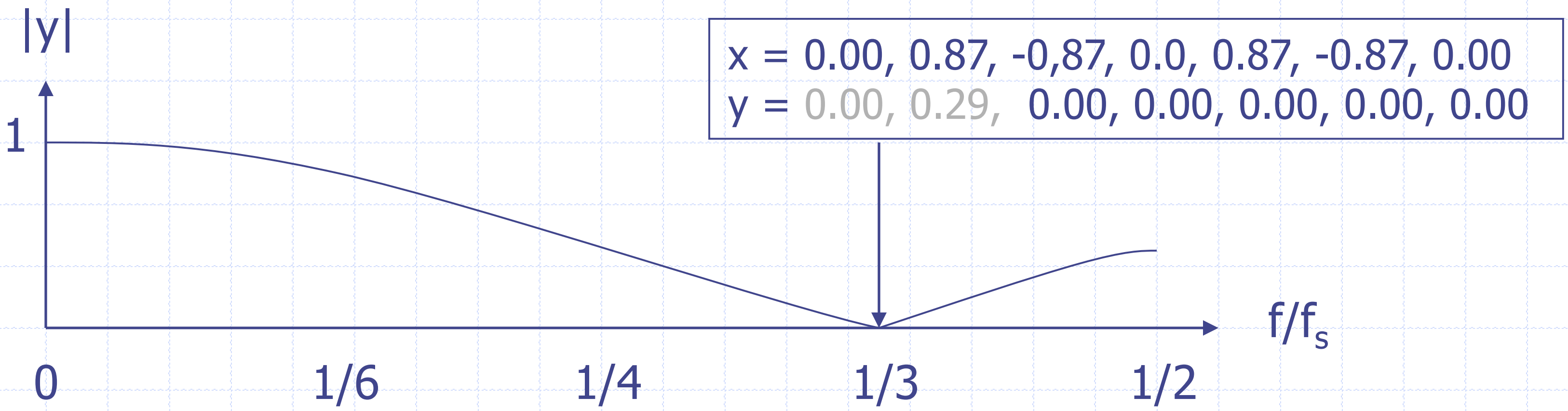
MA filter filters (removes) signals of certain frequency:

Frequency Behavior MA

lower frequency x: amplitude y = 0.77
x = 0.00, 0.33, 0.66, 1.00, 0.66, 0.33, 0.00, -0.33, -0.66, -1.00, -0.66, -0.33, 0.00
y = 0.00, 0.11, 0.33, 0.66, 0.77, 0.66, 0.33, 0.00, -0.33, -0.66, -0.77, -0.66, -0.33

higher frequency x: amplitude y = 0.33
x = 0.00, 1.00, 0.00, -1.00, 0.00, 1.00, 0.00, -1.00, 0.00, 1.00, 0.00, -1.00, 0.00
y = 0.00, 0.33, 0.33, 0.00, -0.33, 0.00, 0.33, 0.00, -0.33, 0.00, 0.33, 0.00, -0.33

transient ↔ steady-state →



x = 0.00, 0.87, -0,87, 0.0, 0.87, -0.87, 0.00
y = 0.00, 0.29, 0.00, 0.00, 0.00, 0.00, 0.00

Outline

- Introduction
- Z Transform
- FIR Filters
- IIR Filters
- Fixed-point Implementation
- Kalman Filter

Analysis: Z Transform

- We can numerically evaluate frequency behavior (see C programs)
- Rather analyze frequency behavior through *analytic* means
- For this we introduce Z transformation

- Let x[n] be a signal in the time domain (n)
- The Z transform of x[n] is given by

$$X(z) = \Sigma_n x[n] z^{-n}$$

where z is a complex variable.

- Example:

$x = 0.00, 0.33, 0.66, 1.00, 0.66, ..$

$X = 0 + 0.33z^{-1} + 0.66z^{-2} + z^{-3} + 0.66z^{-4} + \ldots$

Z Transform

- Z transforms make life easy
- Properties of the Z transform:

- Let $y[n] = x[n-1]$ (i.e., signal delayed by 1 sample)

$$Y(z) = z^{-1} X(z)$$

- Example:

$$x = 0.00, 0.33, 0.66, 1.00, 0.66, ..$$
$$X = 0 + 0.33z^{-1} + 0.66z^{-2} + z^{-3} + 0.66z^{-4} + \ldots$$
$$y = 0.00, 0.00, 0.33, 0.66, 1.00, ..$$
$$Y = 0 + 0z^{-1} + 0.33z^{-2} + 0.66z^{-3} + z^{-4} + \ldots$$
$$= z^{-1} X$$

Z Transform

- Other properties of the Z transform:

- Z transform of K a[n] = K A(z)
- Z transform of a[n] + b[n] = A(z) + B(z)

- Example:

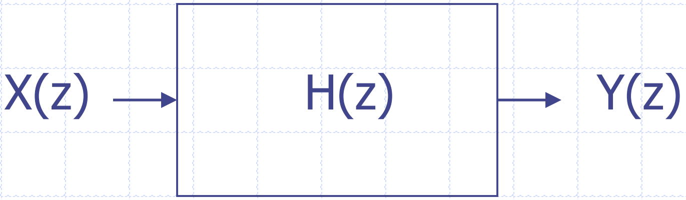
$$x = 0.00, 0.33, 0.66, 1.00, 0.66, ..$$
$$X = 0 + 0.33z^{-1} + 0.66z^{-2} + z^{-3} + 0.66z^{-4} + \ldots$$
$$y = 0.00, 0.66, 1.32, 2.00, 1.32, ..$$
$$Y = 0 + 0.66z^{-1} + 1.32z^{-2} + 2.00z^{-3} + 1.32z^{-4} + \ldots$$
$$= 2\,X$$

Apply Z transform to MA Filter

$y[n] = 1/3\ x[n] + 1/3\ x[n-1] + 1/3\ x[n-2]$

In terms of the Z transform we have:

$$
\begin{aligned}
Y(z) &= 1/3\ X(z) + 1/3\ z^{-1}\ X(z) + 1/3\ z^{-2}\ X(z) \\
&= (1/3 + 1/3\ z^{-1} + 1/3\ z^{-2})\ X(z) \\
&= H(z)\ X(z)
\end{aligned}
$$



- It holds $Y(z) = H(z)\ X(z)$, where $H(z)$ is filter transfer function
- Frequency response of filter can be read from $H(z)$

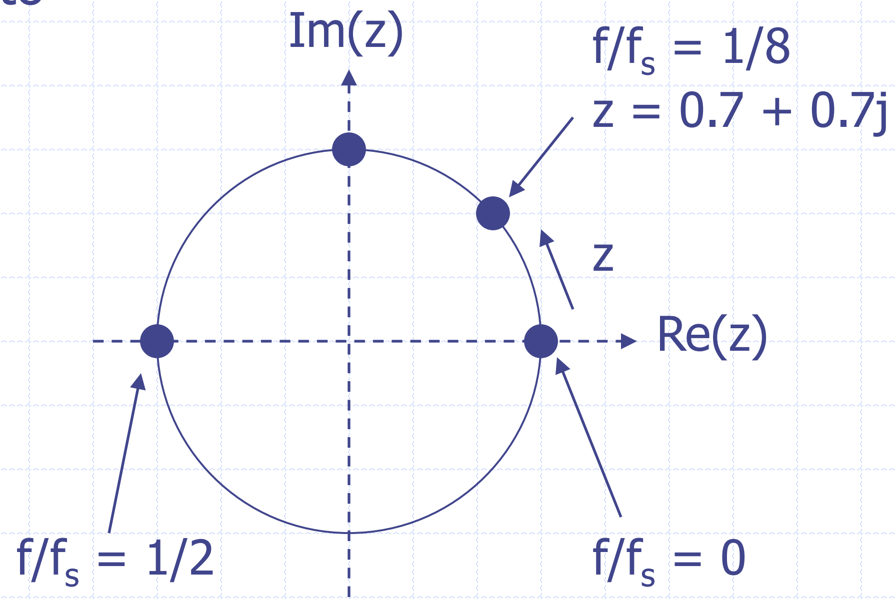
Frequency Response H(z)

H(z) reveals frequency response ($H(f)=H(z)|\ z=e^{j2\pi f}$):
As Y(z) = H(z) X(z), |H(z)| determines *amplification* of X(z)

The variable z is a complex variable and encodes frequency f according to

$$z = e^{j2\pi f}$$
$$= \cos(2\pi f) + j \sin(2\pi f)$$

This corresponds to traversing the unit circle in the complex z plane:



Fourier Interpretation H(z)

Why let z take values $z = e^{j2\pi f}$ where f is frequency?

Recall Z transform of x[n] equals $X(z) = \Sigma_n x[n] z^{-n}$

The Fourier transform of x[n] equals $X(f) = \Sigma_n x[n] e^{-j2\pi nf}$

For a filter with transfer function H(f) its frequency response for a signal with frequency f is |H(f)|

By substituting $z = e^{j2\pi f}$ in H(z) we essentially obtain
the Fourier transform H(f) of which we know |H(f)| is
the frequency response. So let $z = e^{j2\pi f}$ and evaluate |H(z)| !

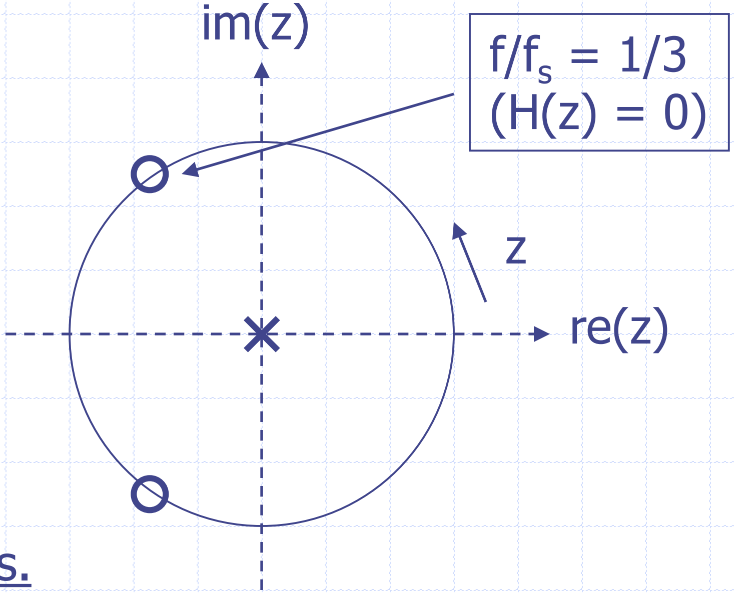
Frequency Response MA Filter

The transfer function of the MA filter is given by:

$$H(z) = (1/3 + 1/3\ z^{-1} + 1/3\ z^{-2})$$
$$= (1/3\ z^2 + 1/3\ z + 1/3)\ /\ z^2 \quad \text{(normalized)}$$

Determine poles and zeros of H(z):

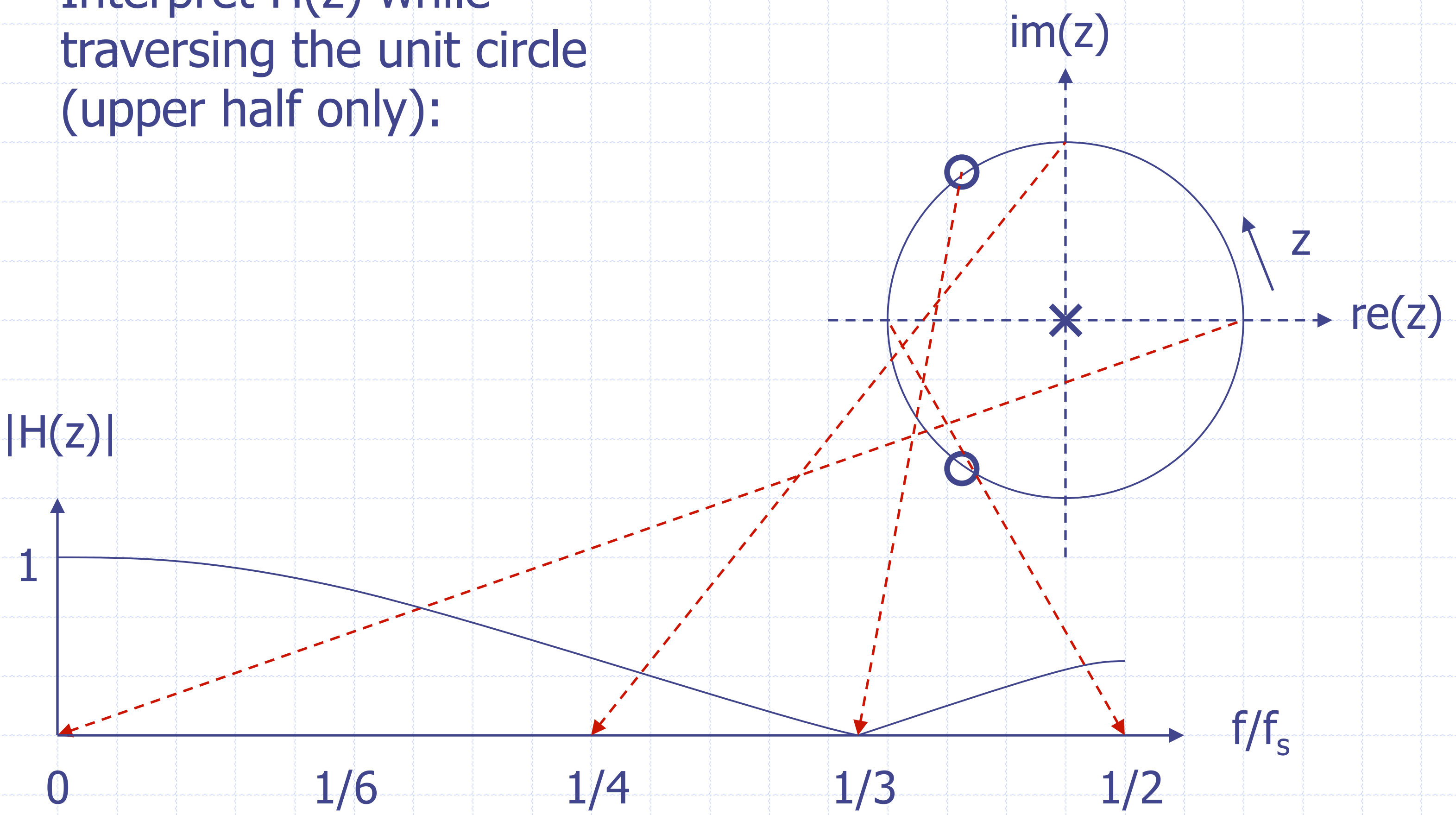
zero (= root of numerator):
$z_1 = -½+½\sqrt{3}j,\ z_2 = -½-½\sqrt{3}j$
$(H(z_{1,2}) = 0)$
pole (= root of denominator):
$z_3,\ z_4 = 0$
$(H(z_{3,4}) = \infty)$



Simply inspect distance z to poles/zeros.

Frequency Response MA Filter

Interpret H(z) while traversing the unit circle (upper half only):



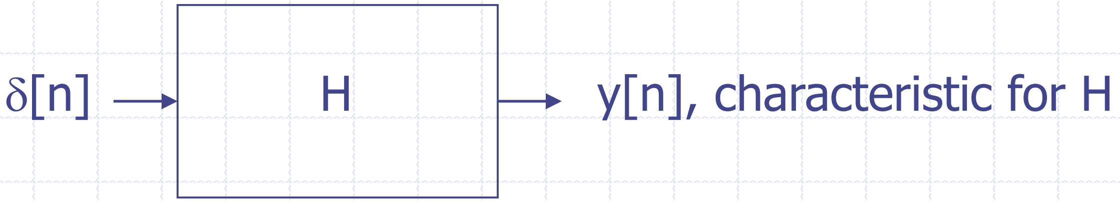
Outline

- Introduction
- Z Transform
- FIR Filters
- IIR Filters
- Fixed-point Implementation
- Kalman Filter

Impulse Response

Impulse signal $\delta[n] = 1, 0, 0, 0, \ldots$ (a spike, Dirac pulse)

Impulse response (IR) of a filter:



$\delta[n] \rightarrow$ H $\rightarrow$ $y[n]$, characteristic for H

MA filter: $y[n] = 1/3\ x[n] + 1/3\ x[n-1] + 1/3\ x[n-2]$

Let $x[n] = \delta[n]$, then $y[n] = 1/3, 1/3, 1/3, 0, 0, 0, \ldots$

Z Transform: $X(z) = 1$, $Y(z) = H(z)\ .\ 1 = H(z) = 1/3 + 1/3z^{-1} + 1/3z^{-2}$

Impulse signal δ reveals $H(z)$ in terms of $h[n]$

Impulse Response

MA filter: h[n] = 1/3, 1/3, 1/3, 0, 0, 0, …

The IR is finite.

Filters defined by

$y[n] = a_0\, x[n] + a_1\, x[n-1] + a_2\, x[n-2] + \ldots$

always have a finite IR and are therefore called FIR filters (the equation is non-recursive in y)

Although any filter can be designed, FIR filters are costly in terms of computation (often many terms needed)

Outline

- Introduction
- Z Transform
- FIR Filters
- IIR Filters
- Fixed-point Implementation
- Kalman Filter

Averaging Filter

Suppose we want to extend MA filter to N terms:

$y[n] = 1/N\ x[n] + 1/N\ x[n-1] + \ldots\ 1/N\ x[n-N-1]$

Suppose we don't want to implement an N-cell FIFO + 2N ops and experiment with the following "short cut":

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

(1st term approximates contents of FIFO after $x[n-N-1]$ has been shifted out, 2nd term is newest sample shifted in)

Let's analyze the frequency response of this filter

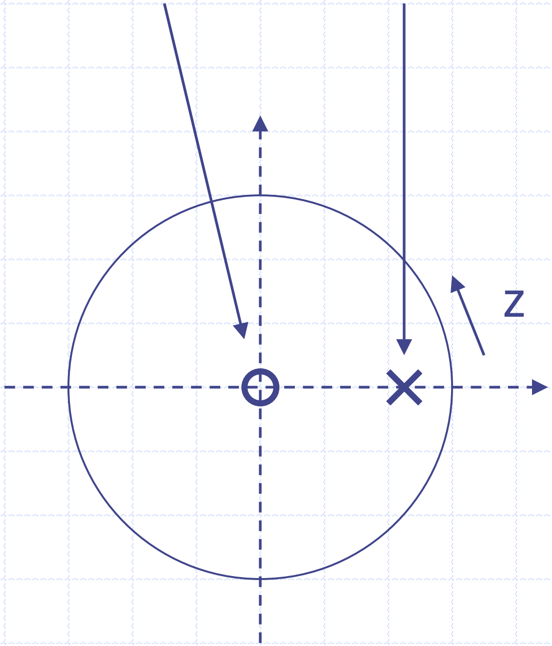
Frequency Response Filter

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

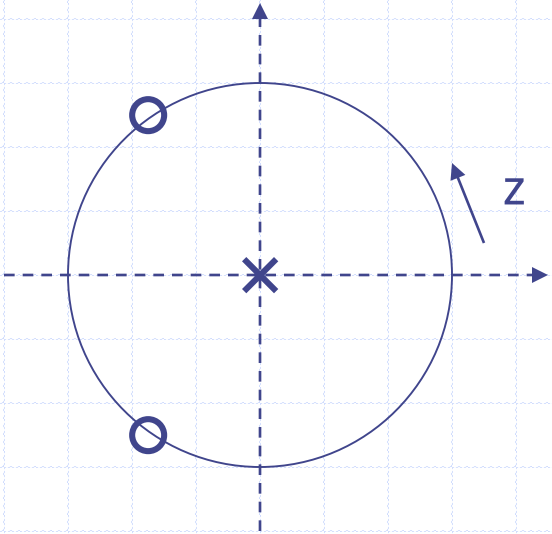
$Y(z) = (N-1)/N\ z^{-1}\ Y(z) + 1/N\ X(z)$

$H(z) = (1/N) / (1 - (N-1)/N\ z^{-1})$

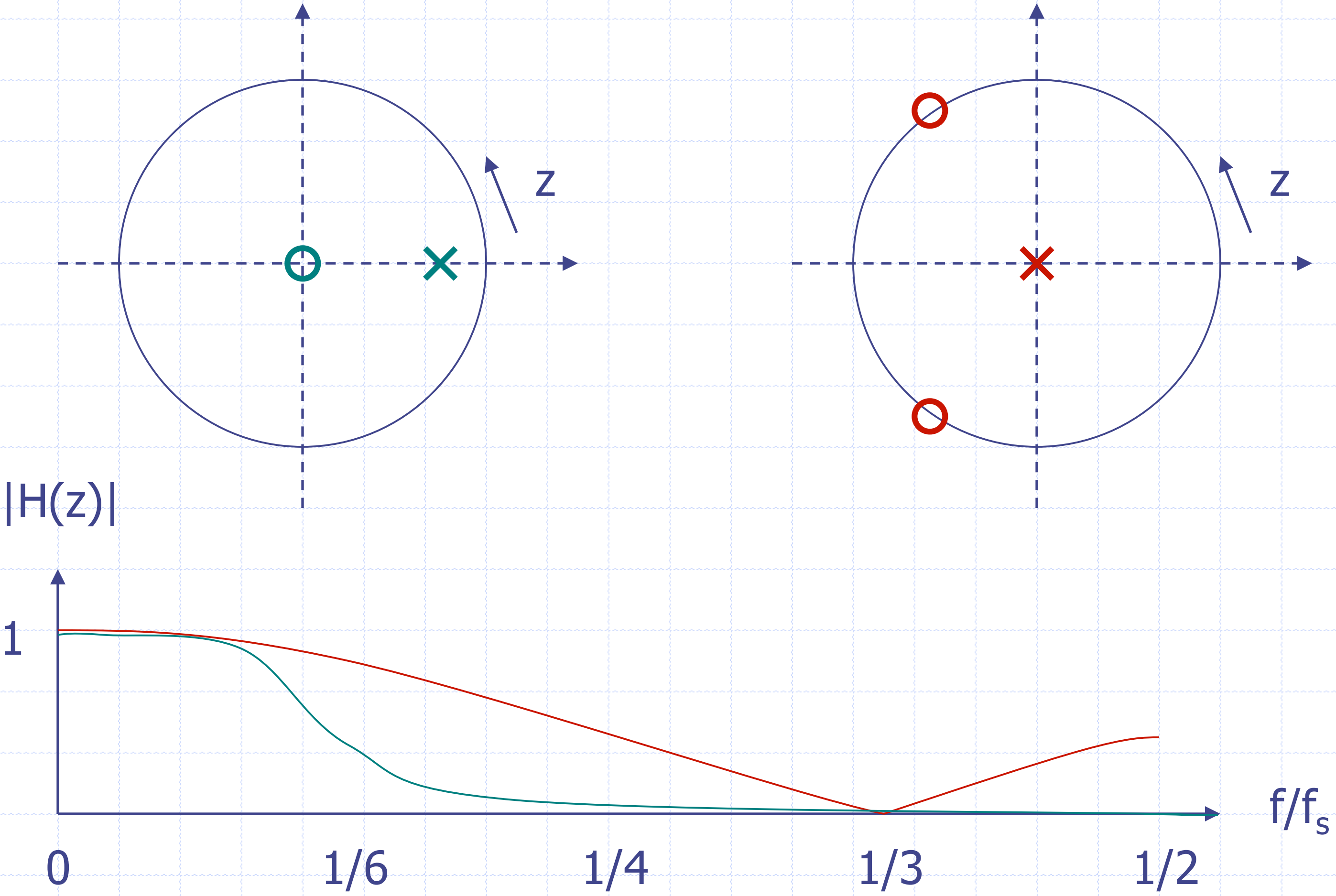
$\quad = (z/N) / (z - (N-1)/N)$



cf. MA filter:



Frequency Response Comparison



Comparison of both Filters

New filter is much more different than perhaps assumed

Pole-zero plot is quite different:
now poles not zero: play an active role

Frequency response is (therefore) more low-pass than MA filter

The closer the pole is to unit circle (larger N),
the sooner is the cut-off (in terms of frequency f),
this generally corresponds to MA filter but this would take large FIFO!

Impulse Response

Filter equation: $y[n] = (N-1)/N \; y[n-1] + 1/N \; x[n]$

IR ($N = 3$): $h[n] = 1/3, (2/3)^1/3, (2/3)^2/3, \ldots, (2/3)^n/3, \ldots$

The IR is infinite.

Filters defined by

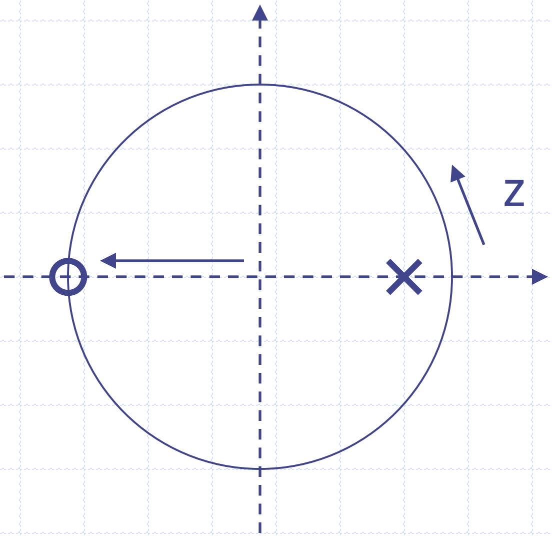
$b_0 \; y[n] + b_1 \; y[n-1] + \ldots = a_0 \; x[n] + a_1 \; x[n-1] + \ldots$

always have an infinite IR and are therefore called IIR filters (the equation is recursive in y)

Filter order determined by # coeficients. Our case: 1st order.

Designing Filters

Looking at the pole-zero plot, the IIR filter can be improved by moving zero to left:
now |H(z)| even becomes zero for $f = f_s/2$
so sharper cut-off.



This plot corresponds to the well-known class of **Butterworth** filters (our case: 1st-order Butterworth):

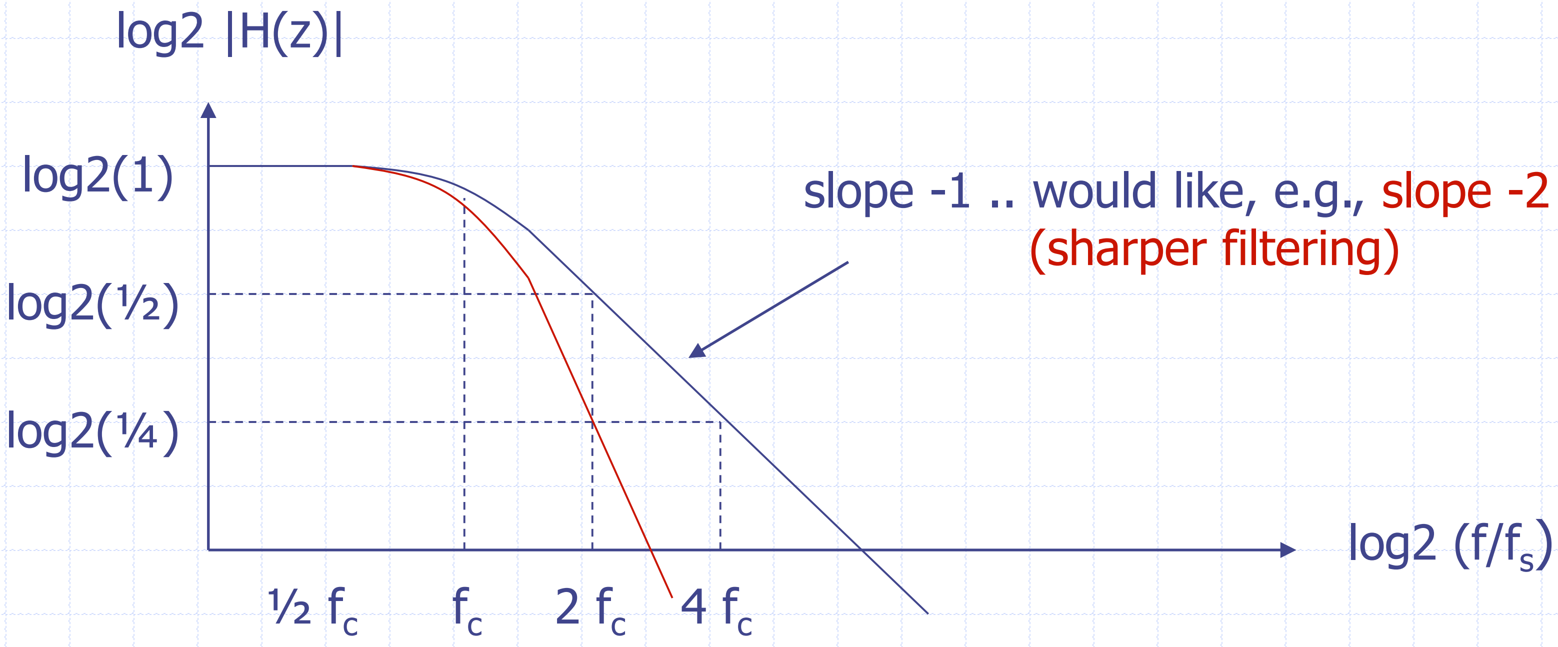
The zero is created by adding x[n-1]:

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

$$H(z) = ((z+1)/2N) \;/\; (z-(N-1)/N)$$

Enhancing Filters

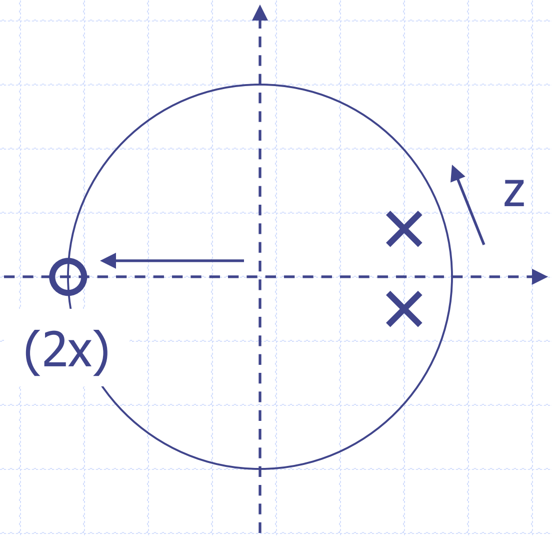
Frequency response 1st-order Butterworth:



slope -1 .. would like, e.g., slope -2 (sharper filtering)

Second-order Butterworth

Looking at the pole-zero plot, the IIR filter can be further improved by introducing more poles & zeros.
now |H(z)| has same cut-off freq f_c
but sharper slope!



Computing h[n] (the a_i and b_i)
is difficult, so use a tool to compute coefficients, given f_s and f_c
(Matlab or Web sites)

Just insert found coefficients in IIR equation

$$b_0\ y[n] + b_1\ y[n-1] + b_2\ y[n-2] = a_0\ x[n] + a_1\ x[n-1] + a_2\ x[n-2]$$

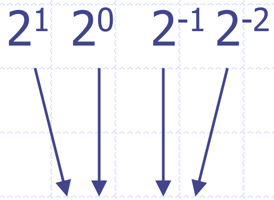
Outline

- Introduction
- Z Transform
- FIR Filters
- IIR Filters
- Fixed-point Implementation
- Kalman Filter

Fixed-point Arithmetic

- Many microcontrollers have no floating-point unit
- Software floating-point often (too) slow
- Need to implement filters in fixed-point arithmetic

2’s-complement bit representation (e.g., 32 bits, 14 bits fraction):



2^1 2^0 2^{-1} 2^{-2}

```
3.75:   000000000000000011  11000000000000
0.02:   000000000000000000  00000101001001
-1.5:   000000000000000001  10000000000000 ^ -1 + 1 =>
        111111111111111110  10000000000000
```

Fixed-point Arithmetic

- Addition, subtraction as usual
- Multiplication: result must be post-processed:
  - make sure intermediate fits in variable! (e.g., 32 bits)
  - shift right by |fraction|

Example multiplication (32 bits, 14 bits fraction):

3.75: 00000000000000001111000000000000 times:
-1.5: 11111111111111111010000000000000 equals:
10100110000000000000000000000000
(value just fits in 32 bits!)
(now shift right by 14 bits and sign-extend):
11111111111111101001100000000000 which is:
-5.625 111111111111111010 01100000000000

Filter Example

- Second-order Butterworth LP Filter $f_c = 10\text{Hz}$, $f_s = 1250\text{Hz}$
- Coefficients:

$a_0 = 0.0006098548$	$a_1 = 2\, a_0$	$a_2 = a_0$
$b_0 = 1$	$b_1 = -1.9289423$	$b_2 = 0.9313817$

Bit representation (e.g., 32 bits, 14 bits fraction):

a[0]	000000000000000000	00000000001010	($a_0 << 14$)
a[1]	000000000000000000	00000000010100	
a[2]	000000000000000000	00000000001010	
b[1]	000000000000000001	11101101110100	^ -1 + 1
b[2]	000000000000000000	11101110011100	

Implementation (high-cost)

```
int mul(int c, int d) {
  int result = c * d;
  return (result >> 14);
}

void filter() {
  y0 = mul(a0,x0) + mul(a1,x1) + mul(a2,x2) -
       mul(b1,y1) - mul(b2,y2);
  x2 = x1; x1 = x0; y2 = y1; y1 = y0;
}
```

Filter Approximation Example

- Second-order Butterworth LP Filter f_c = 10Hz, f_s = 1250Hz
- Coefficients:

$a_0 = 0.0006098548 * 8/10$ $a_1 = 2\ a_0$ $a_2 = a_0$

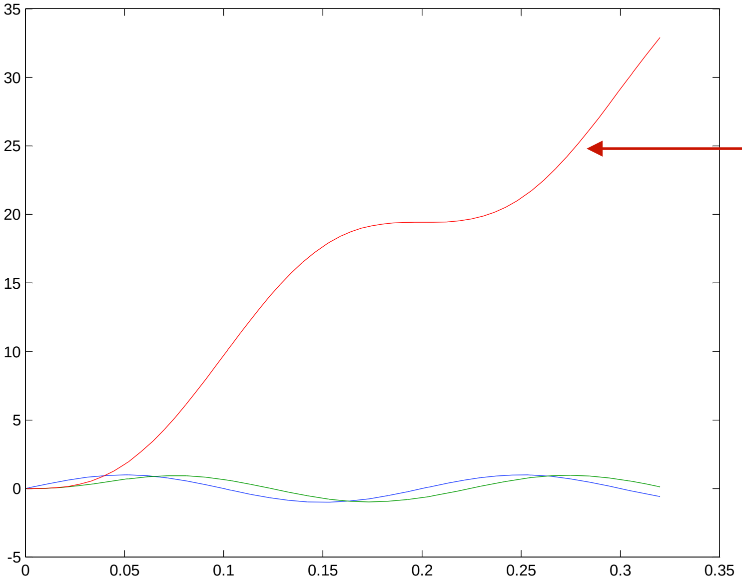
$b_0 = 1$ $b_1 = -2$ $b_2 = 1$

Bit representation (e.g., 32 bits, 14 bits fraction):

a[0]	000000000000000000	00000000001000	(was 10)
a[1]	000000000000000000	00000000010000	(was 20)
a[2]	000000000000000000	00000000001000	(was 10)
-b[1]	000000000000000010	00000000000000	(was 31604)
b[2]	000000000000000001	00000000000000	(was 15260)

Implementation (low-cost)

```
y0 = (x0 << 3) >> 14 + (x1 << 4) >> 14 +
     (x2 << 3) >> 14 + (y1 << 15) >> 14 -
     (y2 << 14) >> 14; // assume compiler optimizes ...
x2 = x1; x1 = x0; y2 = y1; y1 = y0;
```



Approx too coarse
(2nd-order FIR:
a_i, b_i very sensitive!)

Cascade two 1st-order filters

- First-order Butterworth LP Filter f_c = 10Hz, f_s = 1250Hz
- Coefficients:

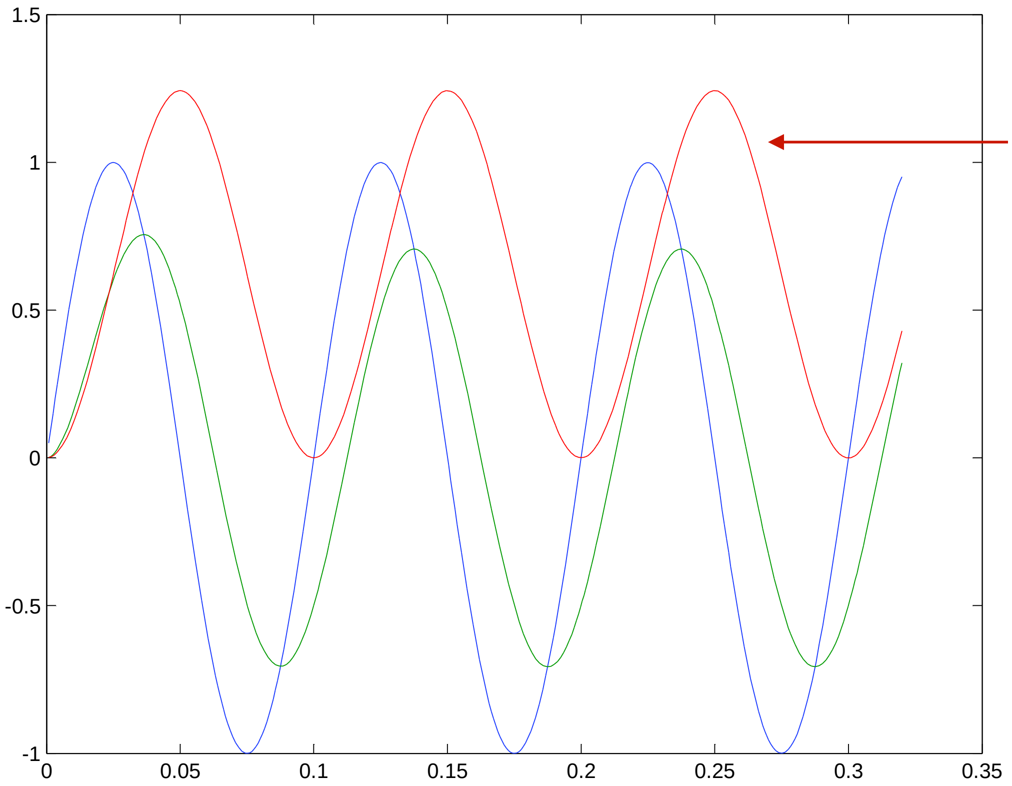
$a_0 = 0.0245221$ $a_1 = a_0$
$b_0 = 1$ $b_1 = -0.95095676$

Bit representation (e.g., 32 bits, 14 bits fraction):

a[0] 000000000000000000 00000110010010 (a_0 << 14)
a[1] 000000000000000000 00000110010010
b[1] 000000000000000000 11110011011100 ^ -1 + 1

Approx: a[0] = 512 (was 402), b[1] = 16384 (was 15580)

Results



Approx bit better
But still bad for very
low frequencies

So add more powers
of two until good approx
(see matlab demo)

Scaling: tips and tricks

- One size fits all? NO!
  - number of bits depends on needed precision (sensor vs. joystick)

  - special case for proportional controller: P * ε
  - $fp_n * fp_n = fp_{2n}$ (overflow! requires an additional shift)
  - $scalar * fp_n = fp_n$ (overflow? no shift needed)
  - $fp_m * fp_n = fp_{m+n}$ (when P can’t be represented as a scalar)

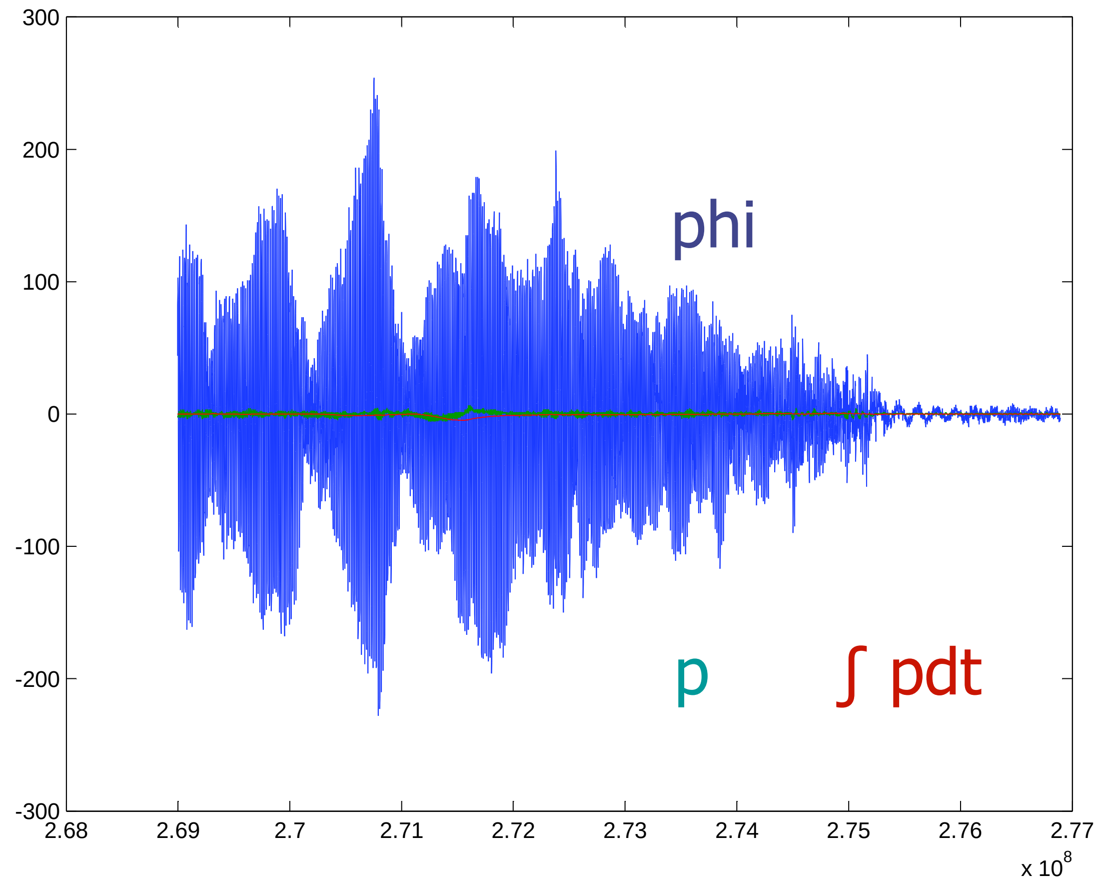
  - document precision for every data type (part of softw arch)

- fp_n to scalar
  - be patient, shift at last instant (when feeding the engines)

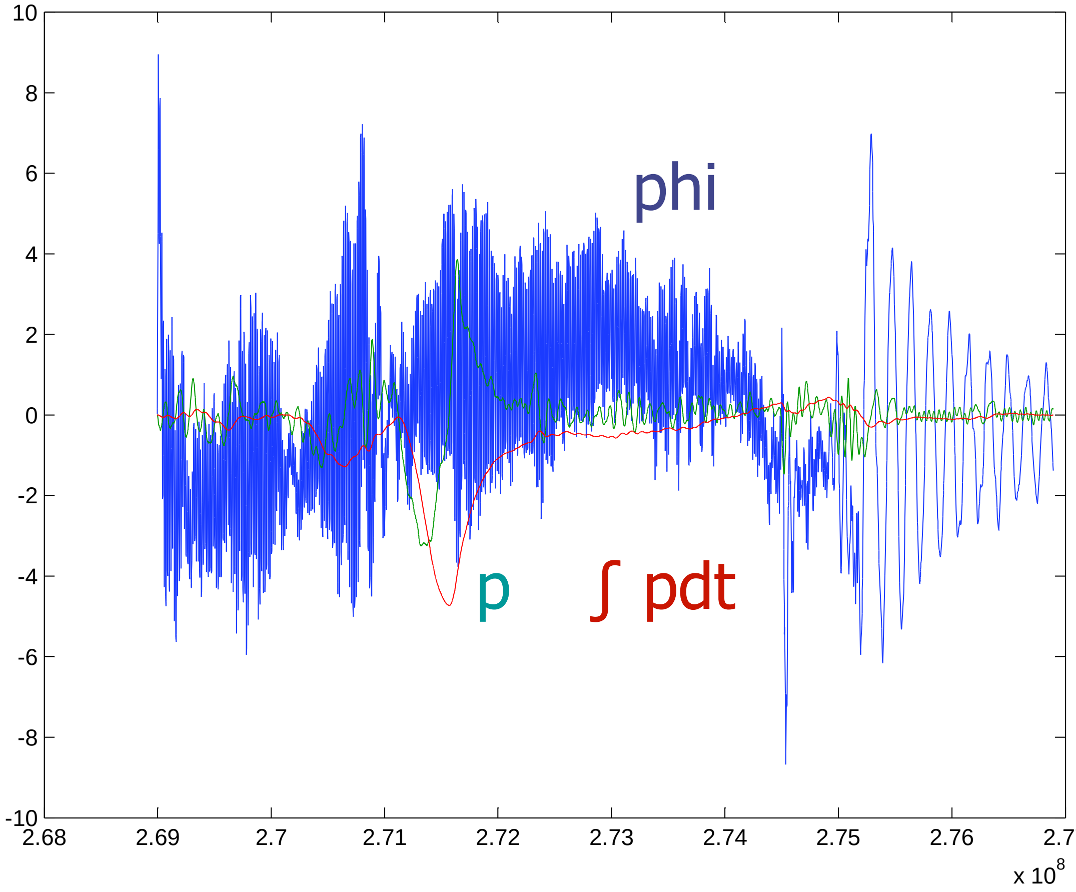
Outline

- Introduction
- Z Transform
- FIR Filters
- IIR Filters
- Fixed-point Implementation
- Kalman Filter

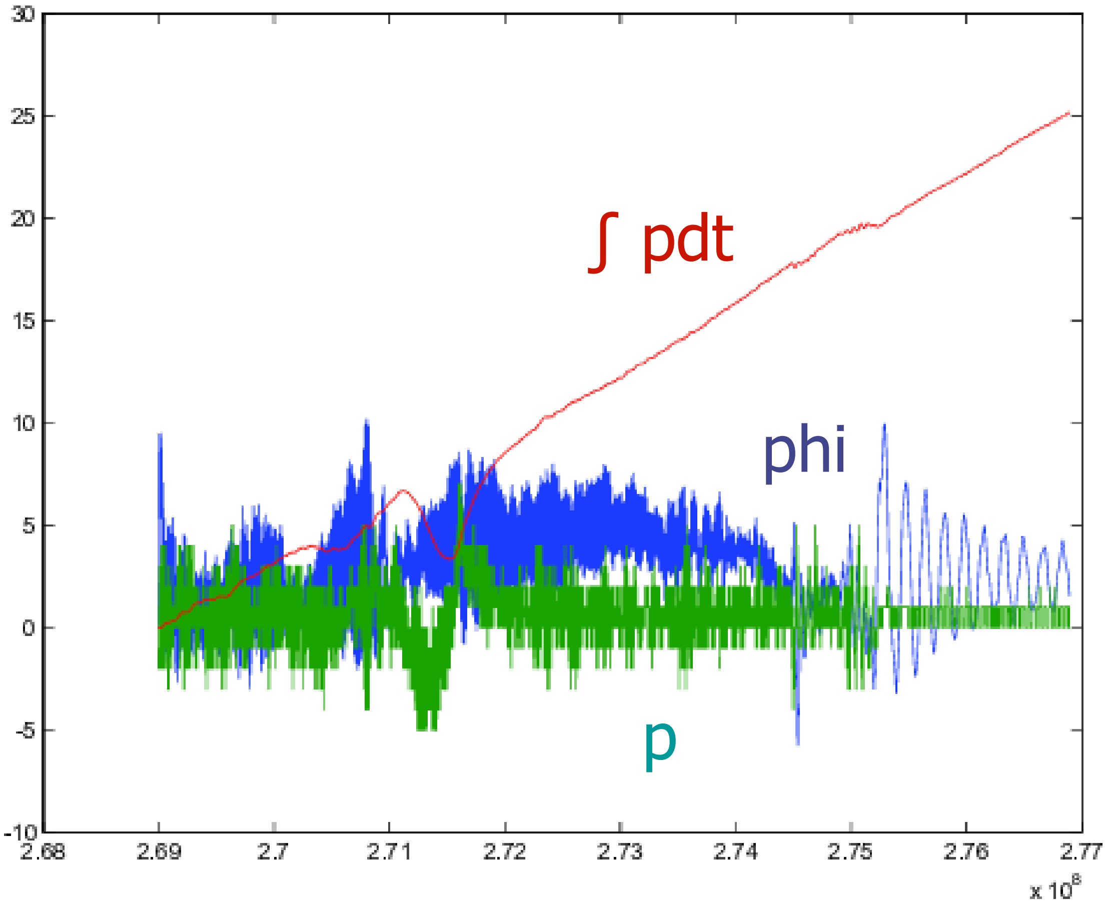
Recall QR Sensor Signals phi, p



After 2nd-order Low-pass (10Hz)



Bias in p: Integration drift in phi



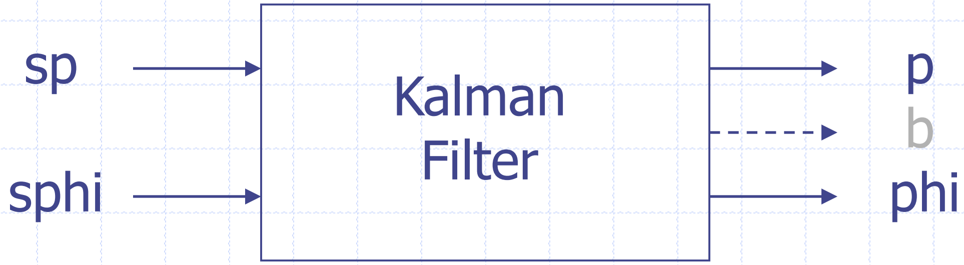
Problem Analysis

- Noise is still considerable
- Still little correlation between (filtered) phi and p
- More aggressive filtering -> more phase delay
- 10 Hz signals already 90 deg phase lag with 2nd-order
- In our particular case we might apply *notch filter*
- In general though, too many noise frequencies
- sphi: negligible drift, too high noise
- sp: low noise, drift -> prohibits integration to phi

- Kalman Filter: combine the best of both worlds!

Kalman Filter (near-hover)

- Sensor Fusing: gyro and accel share same information



- Integrate sp to phi
- *Adjust* integration for sp (drift) bias *b* by comparing phi to sphi, averaged over *long* period (phi ~ constant)
- Return phi, and p (= sp – bias)

Algorithm

- p = sp – b // estimate real p
- phi = phi + p * P2PHI // predict phi
- e = phi – sphi // compare to measured phi
- phi = phi – e / C1 // correct phi to *some* extent
- b = b + (e/P2PHI) / C2 // adjust bias term

- P2PHI: depends on loop freq -> compute/measure
- C1 small: believe sphi ; C1 large: believe sp
- C2 large (typically > 1,000 C1): slow drift

Summary

- DSP is everywhere
- This was merely introduction into the field
- Get a feel for it when applying to QR