D. Richard Brown III
Associate Professor
Worcester Polytechnic Institute
Electrical and Computer Engineering Department
drb@ece.wpi.edu

Lecture 2

ECE4703 REAL-TIME DSP
SAMPLING, QUANTIZATION, REAL-TIME FIR FILTERING

Some Challenges of Real-Time DSP

- Analog to digital conversion
  - Are we sampling fast enough?
  - How much quantization noise have we added to the original analog signal?
  - Are we clipping?
  - ADC non-idealities like non-linear response, etc.
- Digital to analog conversion
  - How much distortion is added by the reconstruction filter?
  - DAC non-idealities like non-linear response, etc.
- DSP
  - Are we running in real time?
  - Do we have enough memory?
  - Distortion caused by digital processing, e.g. overflow, underflow, fixed point effects, etc.

Analog To Digital Conversion

- An ADC performs two functions:
  - sampling: convert a continuous-time (CT) signal to a discrete-time (DT) signal

$$x[k] = x_a(k \cdot T_s + \eta)$$

  - quantization: convert a continuous-valued (CV) signal to a discrete-valued (DV) signal

$$x_q[k] = \text{quant}\{x[k], N\}$$

WPI

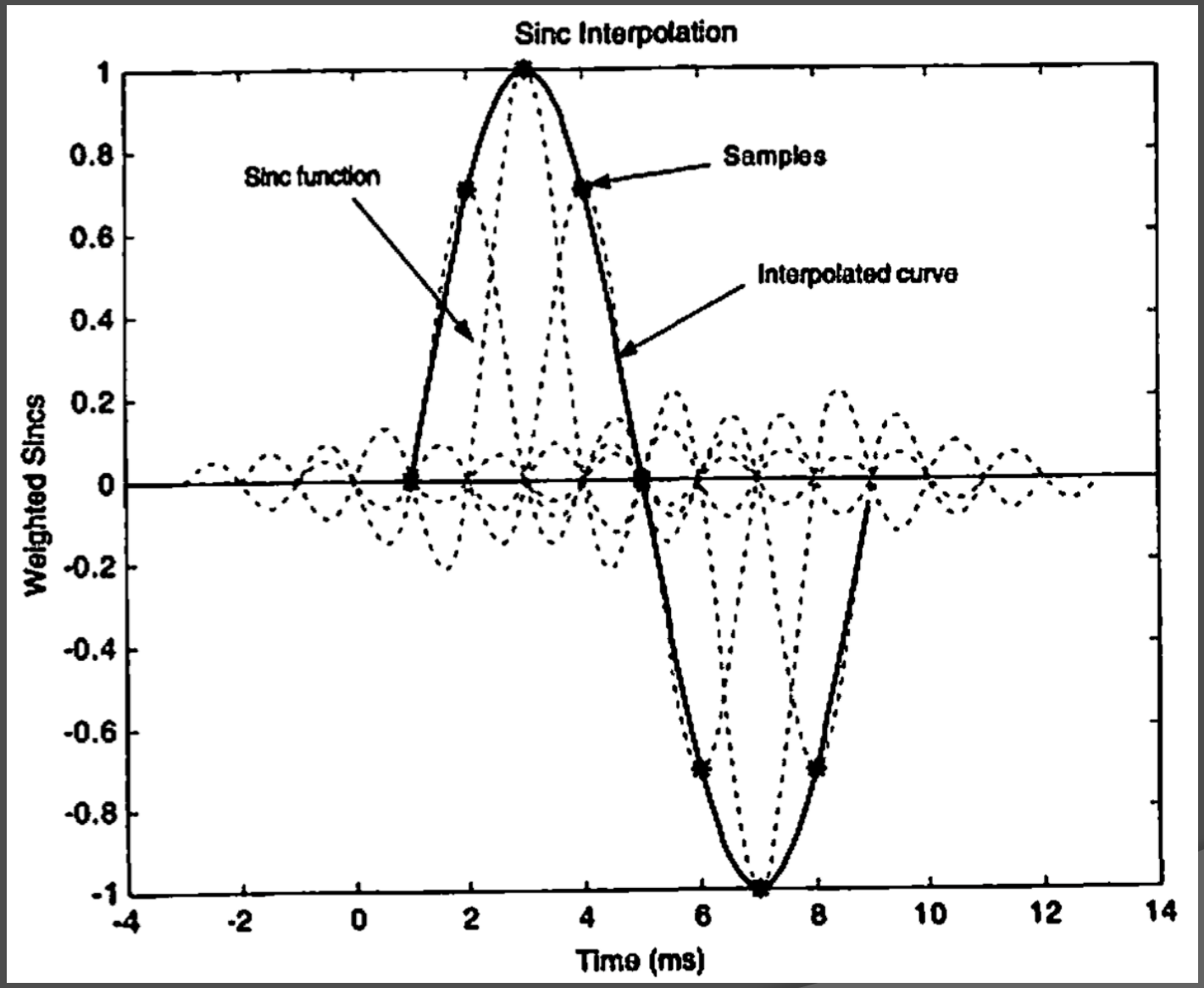
ADC Sampling (CT ➔ DT)

- Recall Nyquist's sampling theorem (ECE2312): *A bandlimited CT signal with maximum frequency B Hz can be uniquely recovered from its samples only if the sampling frequency f_s >= 2B samples per second*
- Reconstruction formula (DT ➔ CT, performed by DAC):

$$x_r(t) = \sum_{k=-\infty}^{\infty} x[k] \frac{\sin(\pi(t - kT_s)/T_s)}{\pi(t - kT_s)/T_s}$$

$x_r(t) = x_a(t)$ if f_s >= 2B (see ECE2312 textbook for proof).

DAC Sinc Reconstruction (Kehtarnavaz Figure 2-17)

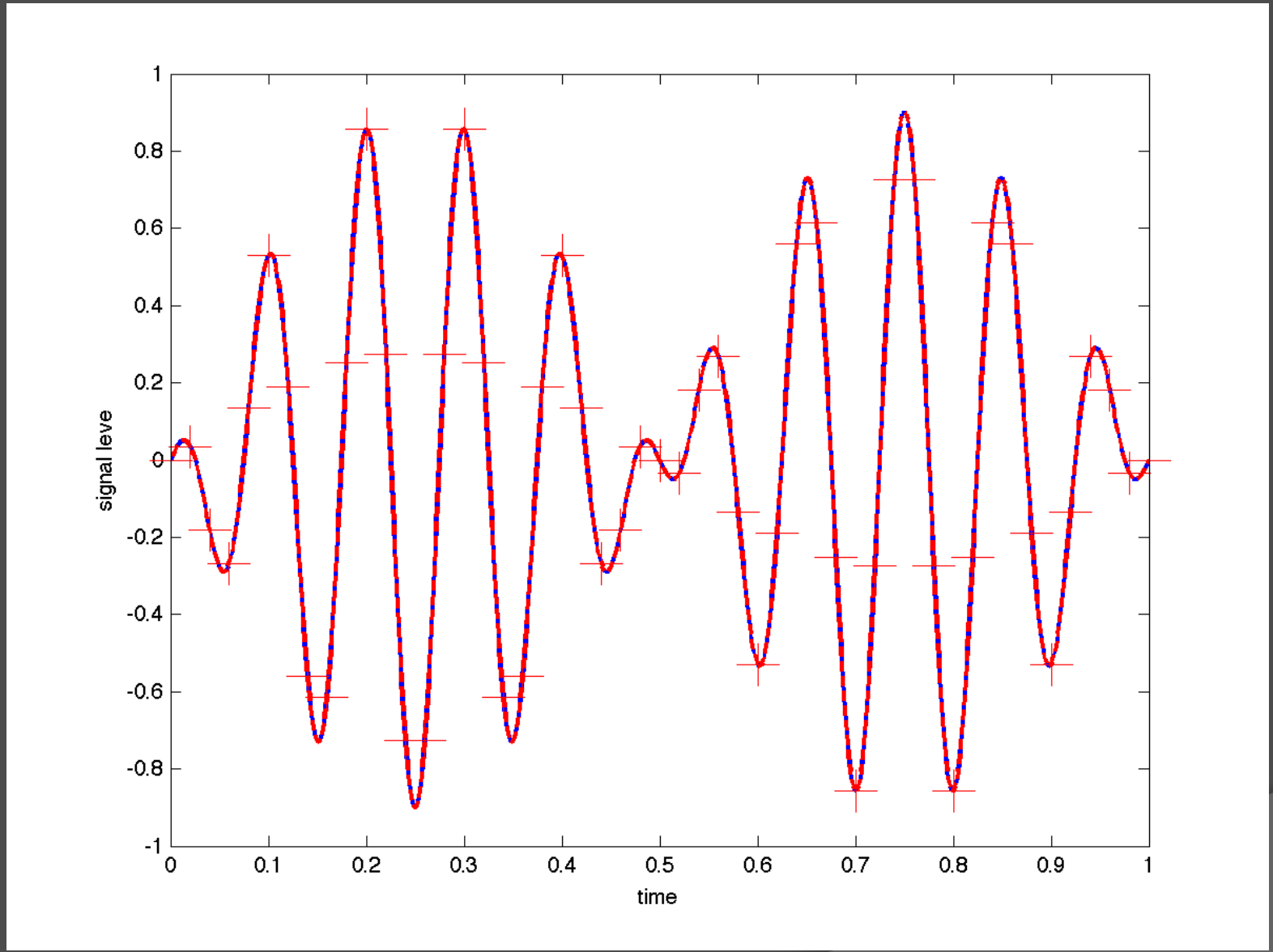


Sampling Example

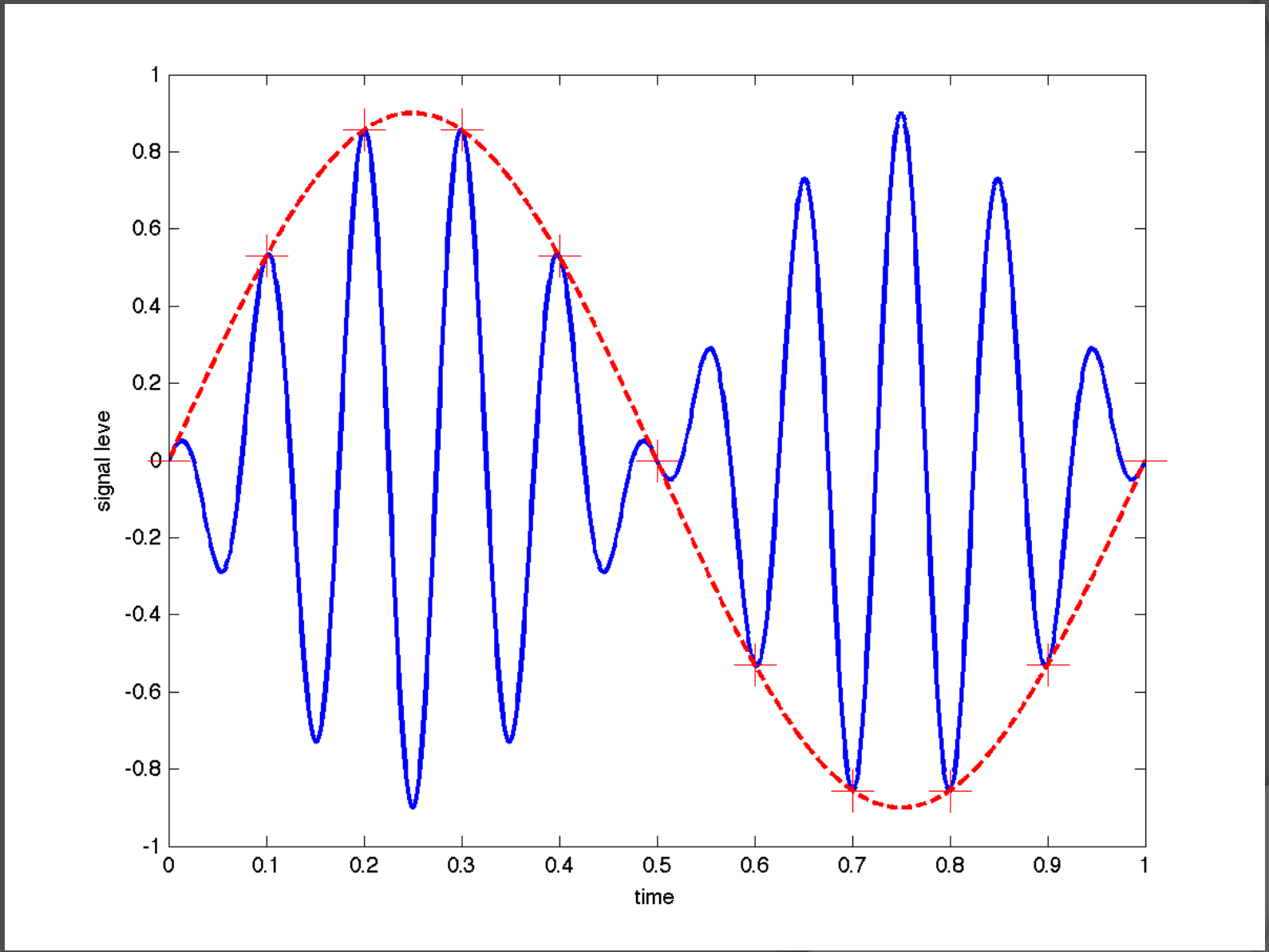
$$x_a(t) = \sin(2\pi \cdot t) \cdot \cos(2\pi \cdot 10 \cdot t)$$

- What is the minimum sampling frequency to allow for exact recovery of the original analog signal from its samples?

Sampling Example: No Aliasing (f_s=50)



Sampling Example: Aliasing (f_s=10)



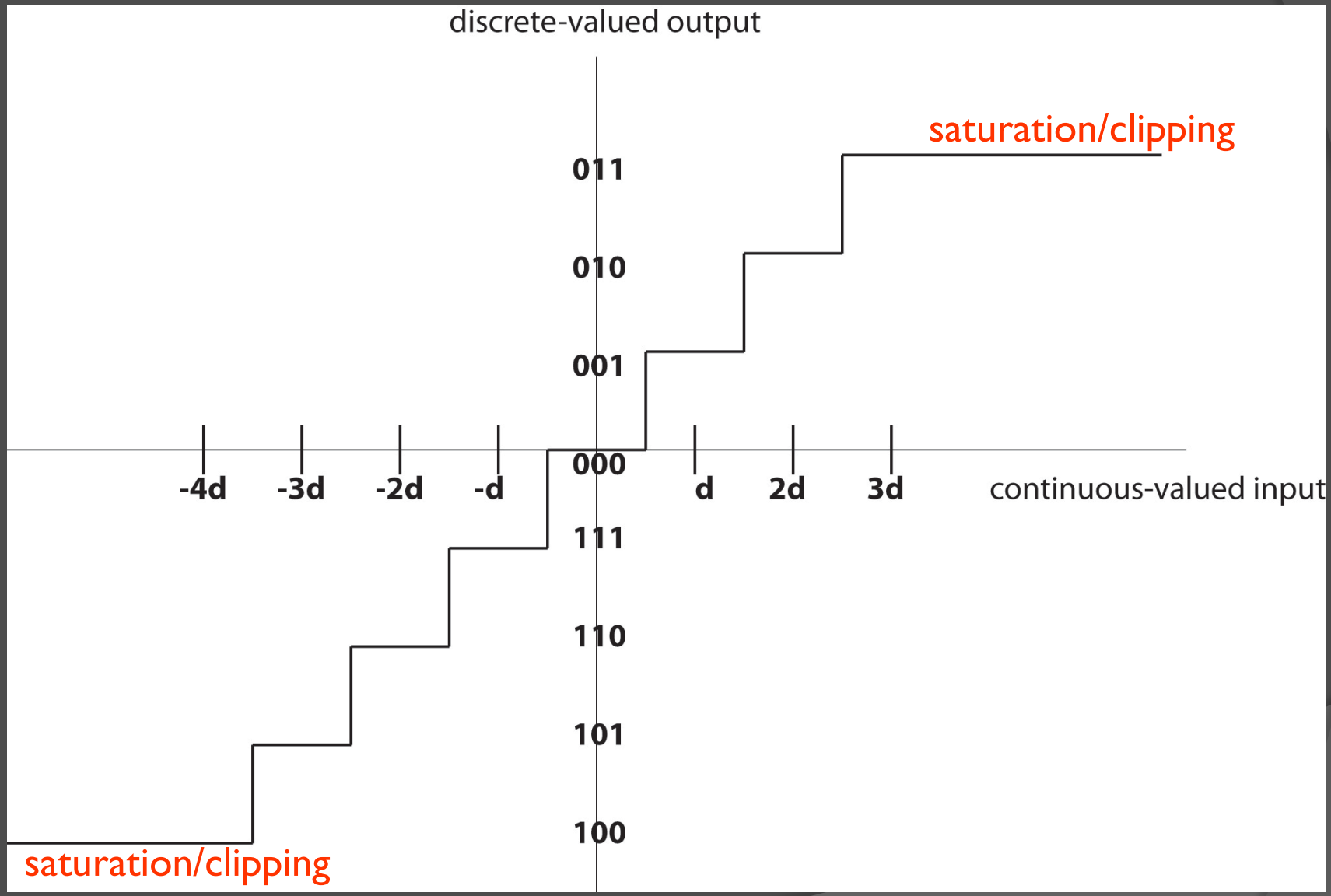
Aliasing Audio Examples

- Please see file aqc.m on course website

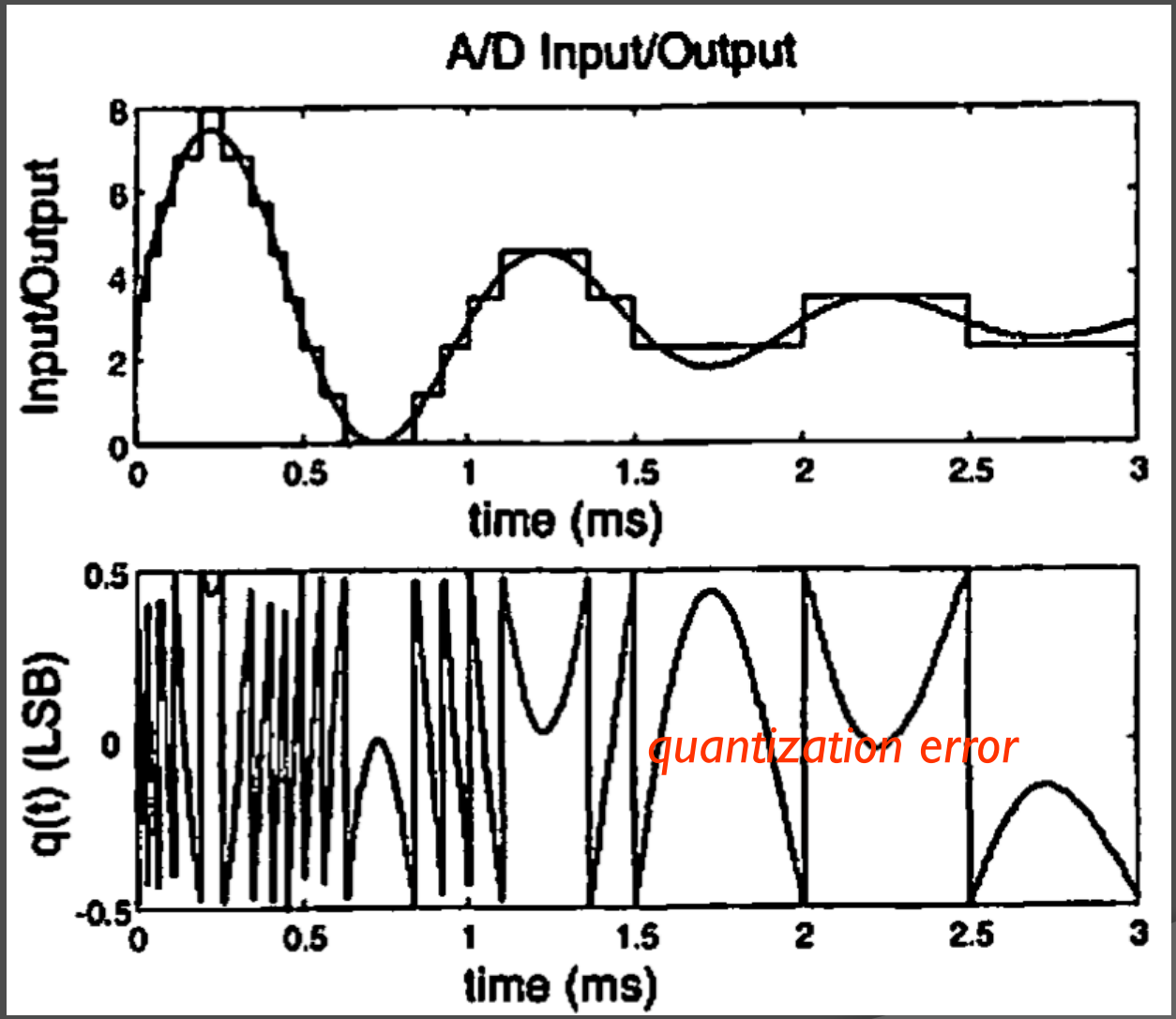
ADC Quantization (CV ➔ DV)

- An N-bit quantizer converts a continuous valued (CV) signal to a discrete valued (DV) signal with 2^N discrete values
- Remarks:
  - Unlike sampling, quantization always causes irreversible distortion of the signal
  - Two types of distortion:
    - Saturation/clipping
    - Quantization error
  - In normal cases with no clipping, increasing the number of bits (N) typically decreases the distortion caused by quantization

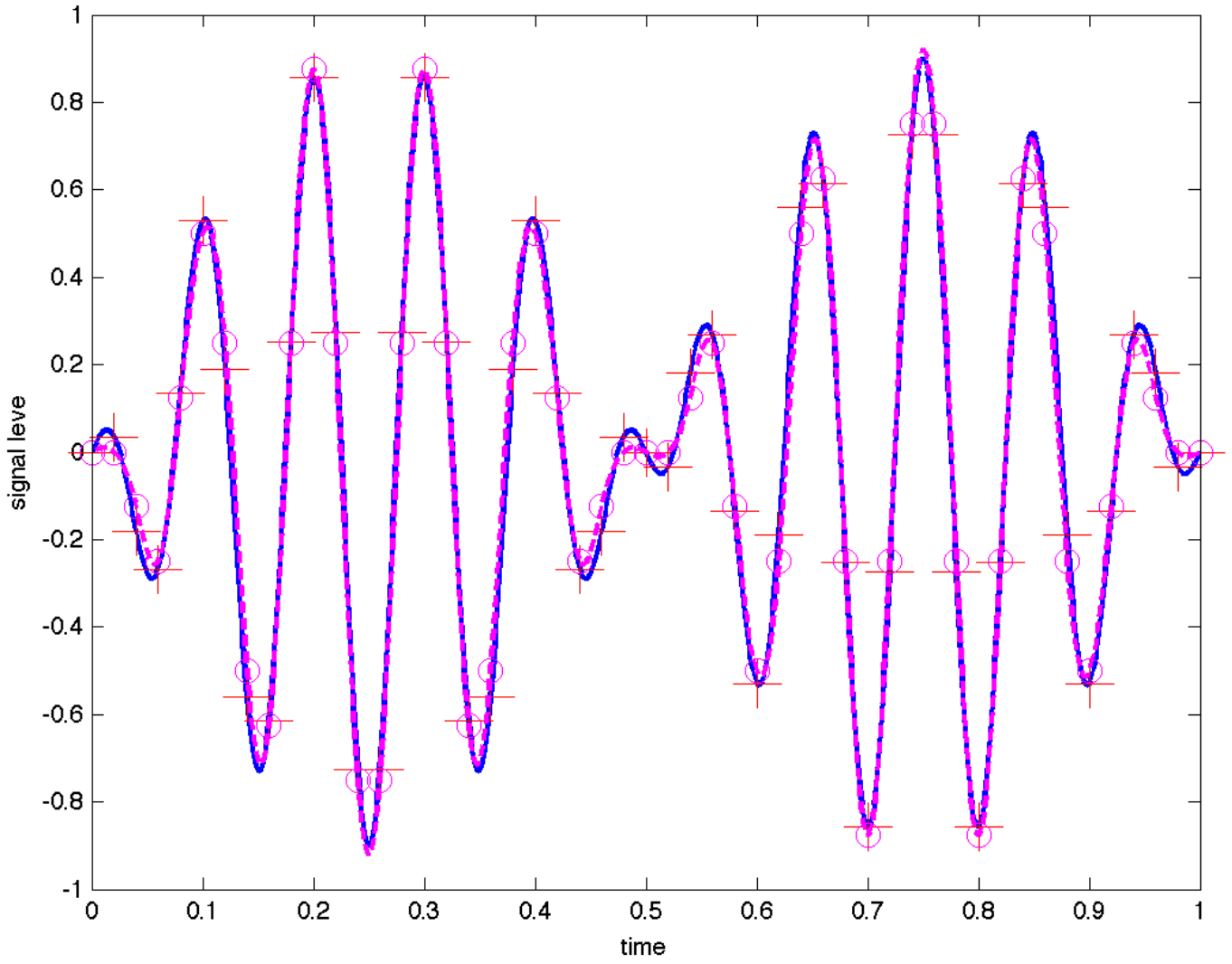
3-bit Ideal Quantization



3-bit Ideal Quantization (Kehtarnavaz Figure 2-15)



Quantizer + Reconstruction Example (N=4, f_s=50)



Signal to Noise Ratio of Quantization

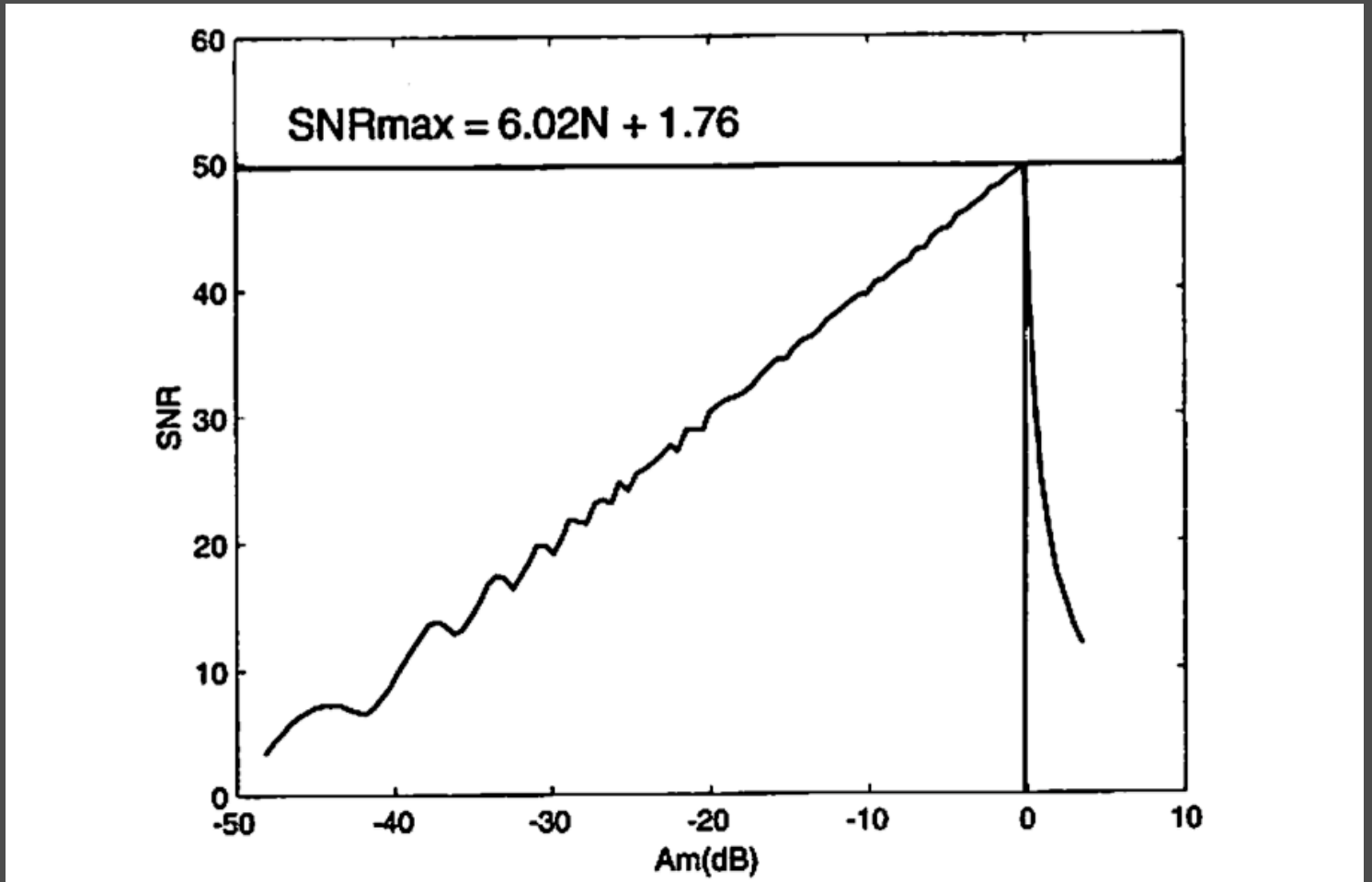


Figure 2-17: Signal-to-noise ratio of an ideal 8-bit A/D converter.

Bottom line: Best SNR is achieved when analog input signal amplitude is as large as possible without saturation/clipping.

Quantization in Matlab

- Matlab variables are typically 64-bit double-precision floating point. We often refer to this as "infinite precision".
- One way to quantize vectors Matlab:
  - First check for saturation:

```
vref = 1;
i1 = find(x>vref*(2^(N-1)-1)/(2^(N-1)));
x(i1) = vref*(2^(N-1)-1)/(2^(N-1));
i2 = find(x<-vref);
x(i2) = -vref;
```

  - Then perform quantization:

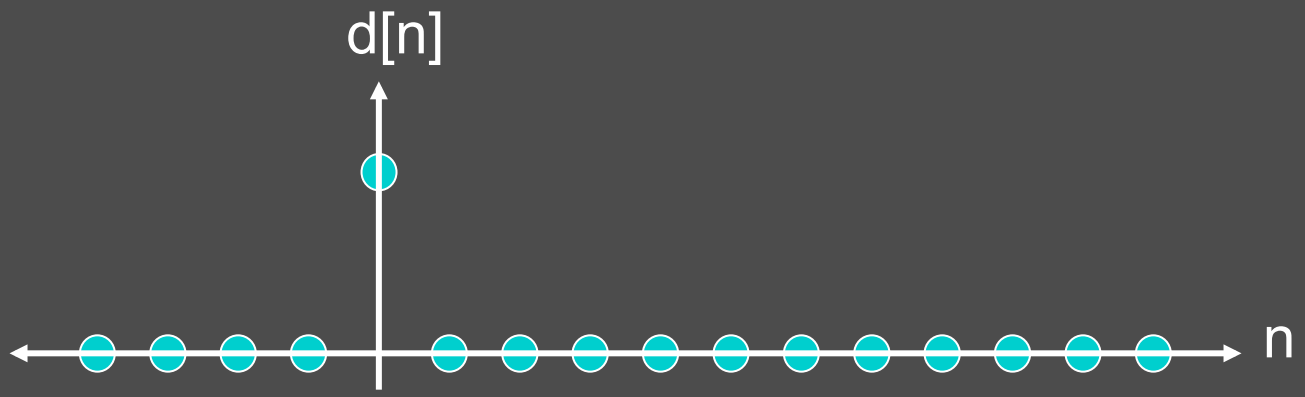
```
xq = round((x/vref)*2^(N-1))*vref/(2^(N-1));
```

  - You can also compute quantization error

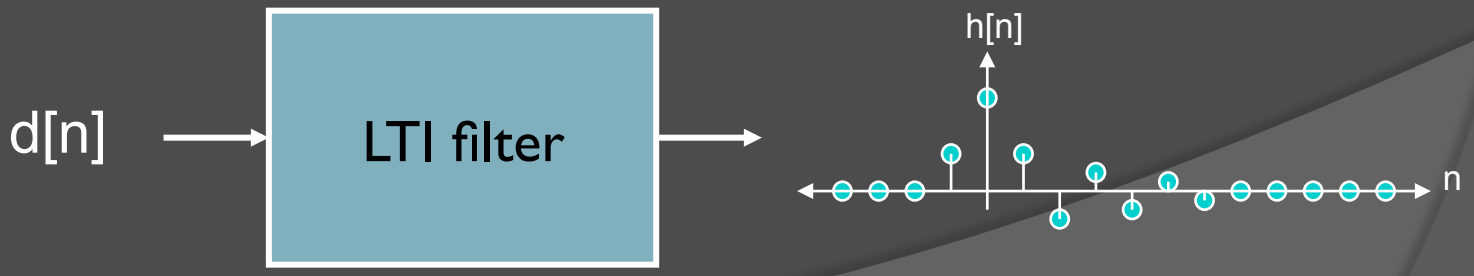
```
equant = x-xq;
```

Signals Review: Impulse Response

- Definition: A discrete time impulse function, d[n], is defined as: d[n] = 1 if n=0, d[n] = 0 otherwise.



- Definition: The "impulse response" of a linear time invariant filter is the output that occurs if the input is d[n].



Finite Impulse Response (FIR) Filtering – Basics

- Definition: A filter is FIR if there exists $N<\infty$ such that the filter's impulse response $h[n]=0$ for all $n>N$.
- FIR filters are frequently used in real-time DSP systems
  - Simple to implement
  - Guaranteed to be stable
  - Can have nice properties like linear phase
- Input/output relationship

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n-m]$$

x = input, ***y*** = output, ***h*** = impulse response (aka "filter coefficients")
M = # of filter coefficients

Finite Impulse Response (FIR) Filtering – More Basics

- Transfer function (useful for what?)

$$H(z) = \sum_{m=0}^{M-1} z^{-m} h[m]$$

- Frequency response (useful for what?)

$$H(e^{j\omega}) = \sum_{m=0}^{M-1} e^{-j\omega m} h[m]$$

Implementation of FIR Filters

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n-m]$$

- If everything is "infinite precision", then there isn't too much to worry about (except real-time considerations)
- Finite precision raises some issues:
  - Precision:
    - How is the input signal quantized?
    - How is the output signal quantized?
    - How are the filter coefficients quantized?
    - How are intermediate results (products, sums) quantized/stored?
  - "Realization Structure"
    - In what order should we do the calculations?

Actual performance can be significantly affected by these choices.

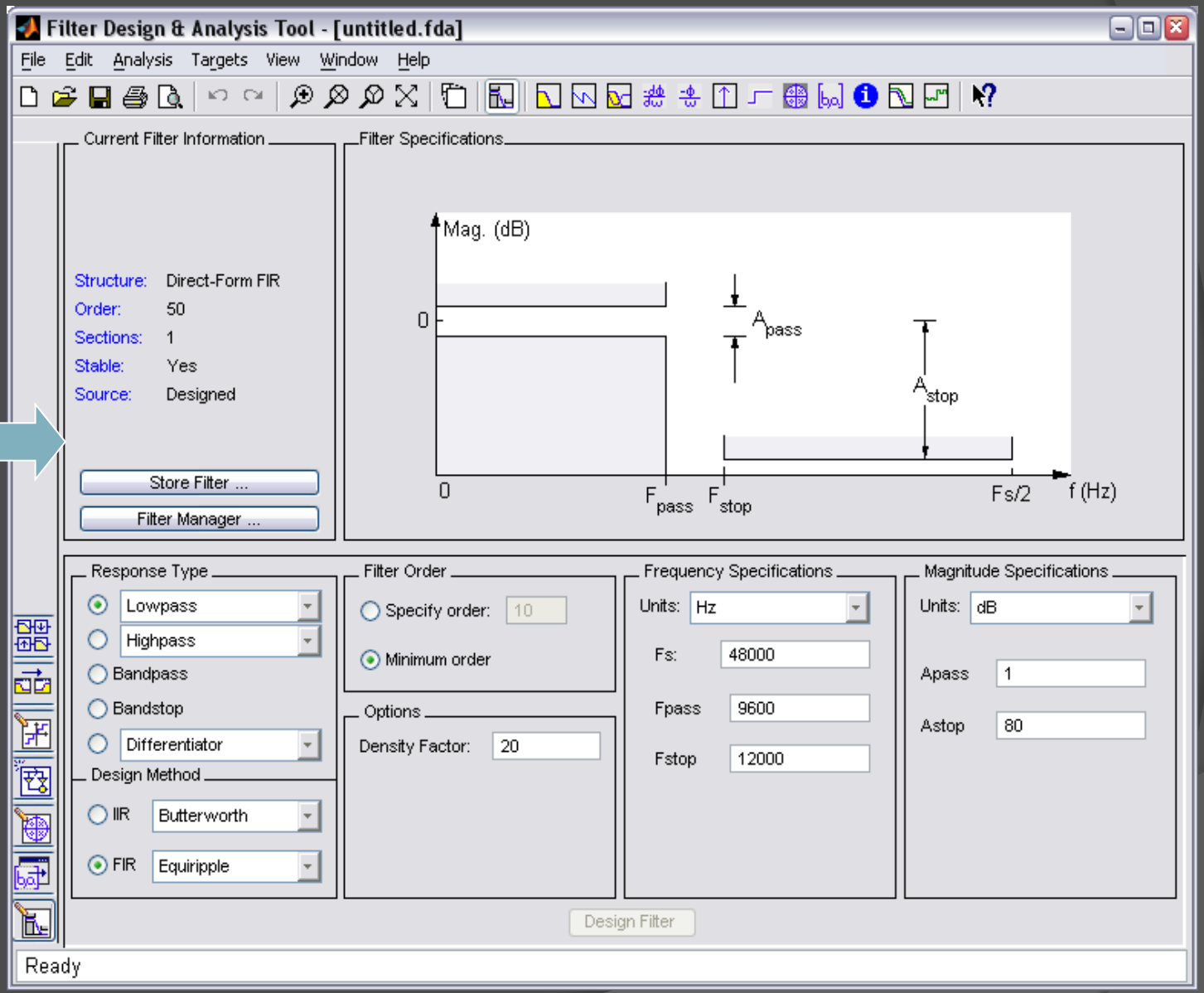
FIR filtering is usually less sensitive to these choices than IIR filtering because there is no feedback.

Typical Procedure for Designing and Implementing FIR Filters

1. Design filter — ***Matlab***
   - Type: low pass, high pass, band pass, band stop, ...
   - Filter order M
   - Desired frequency response
2. Decide on a realization structure
3. Decide how coefficients will be quantized.
4. Compute coefficients
5. Decide how everything else will be quantized (input samples, output samples, products, and sums) — ***CCS***
6. Write code to realize filter (based on step 2)
7. Test filter and compare to theoretical expectations

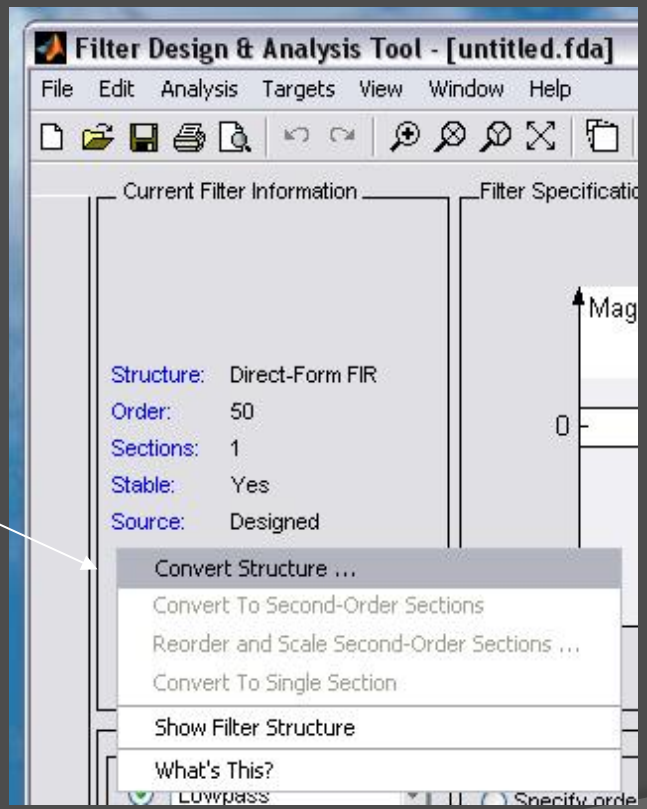
Tools for Designing FIR Filters

>> fdatool

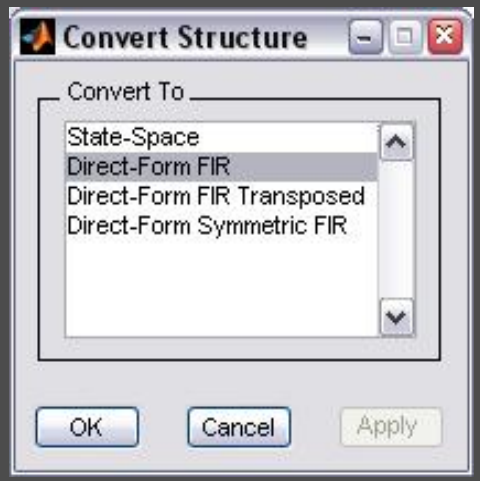


Filter Realization Structures

- Filter realization structure specifies how past calculations are stored and the order in which calculations are performed.
- Lots of different structures available
  - Direct form I, direct form II, transposed forms, cascade, parallel, lattice, …
  - Choice of structure affects computational complexity and how quantization errors are manifested through the filter

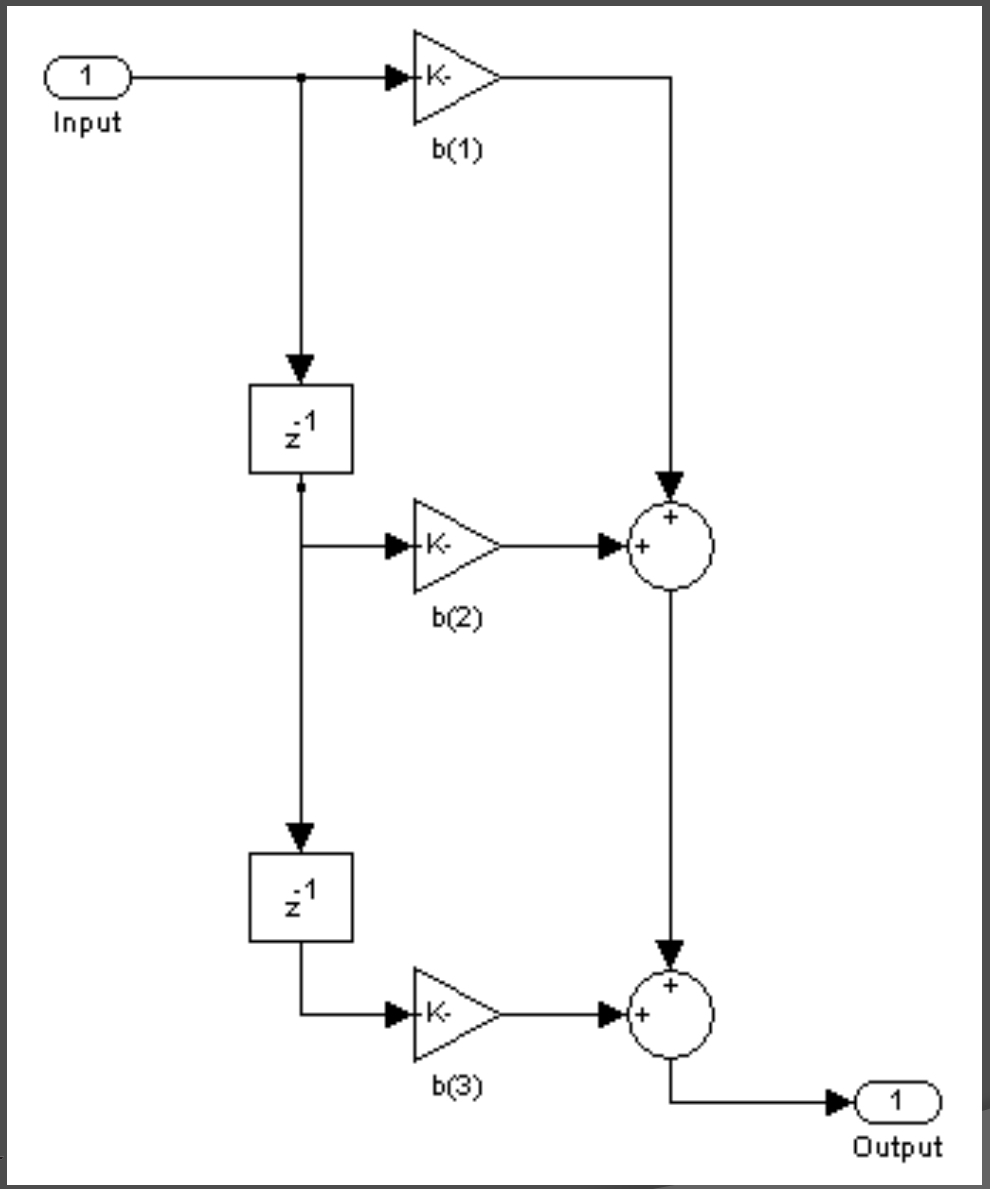


right click in this pane



Focus on "Direct form" for now. We'll discuss other options when we look at IIR filtering later.

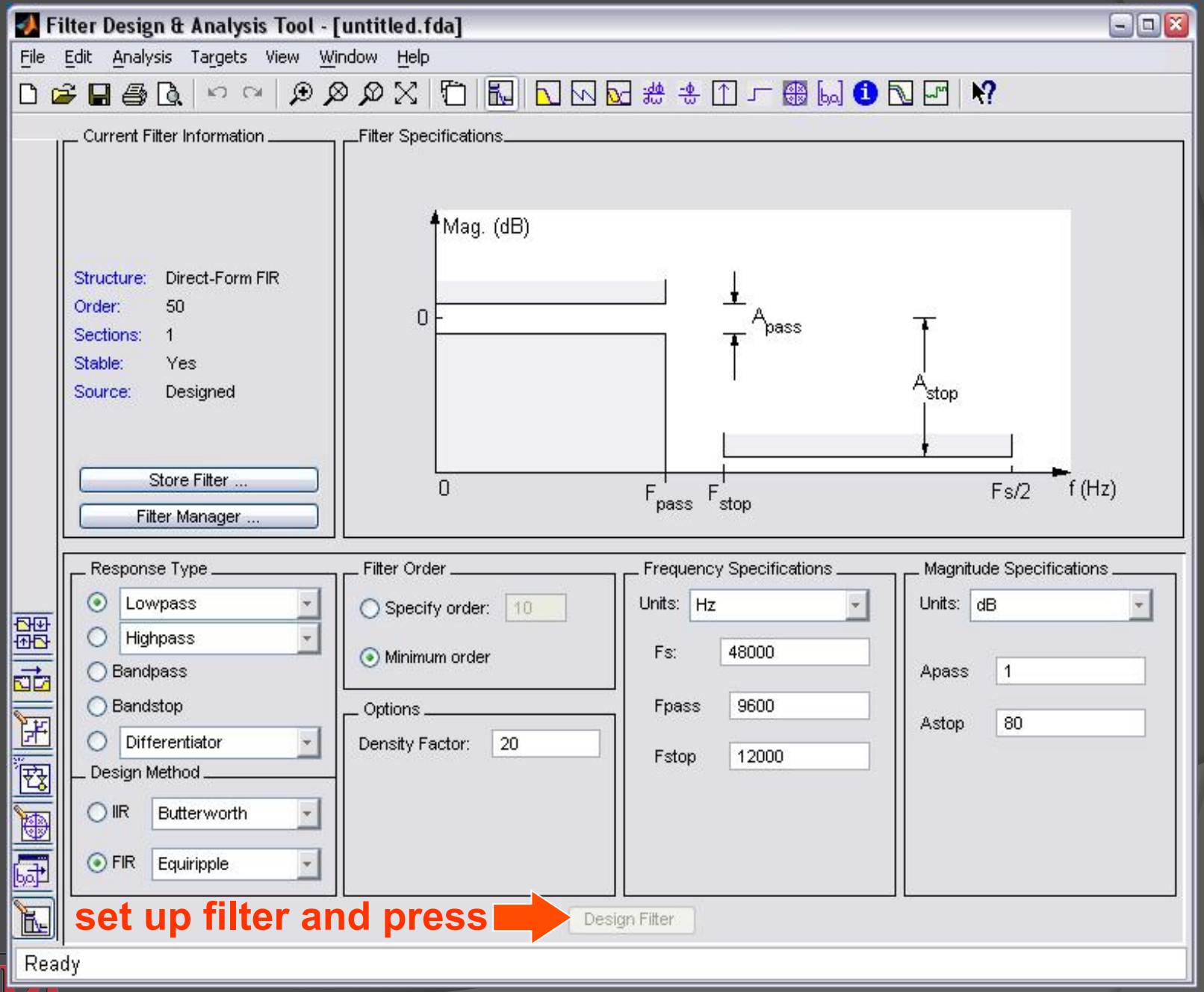
Direct Form I Filter Structure



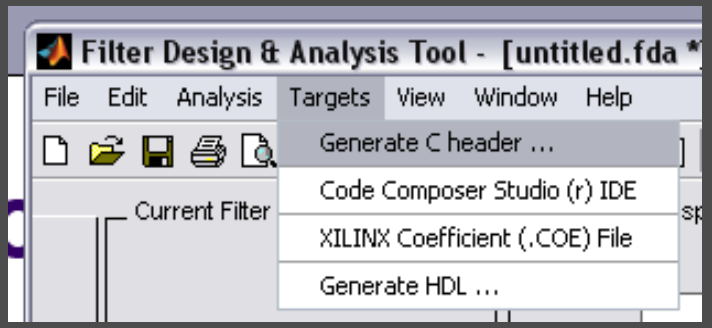
Just a pictorial depiction of convolution.

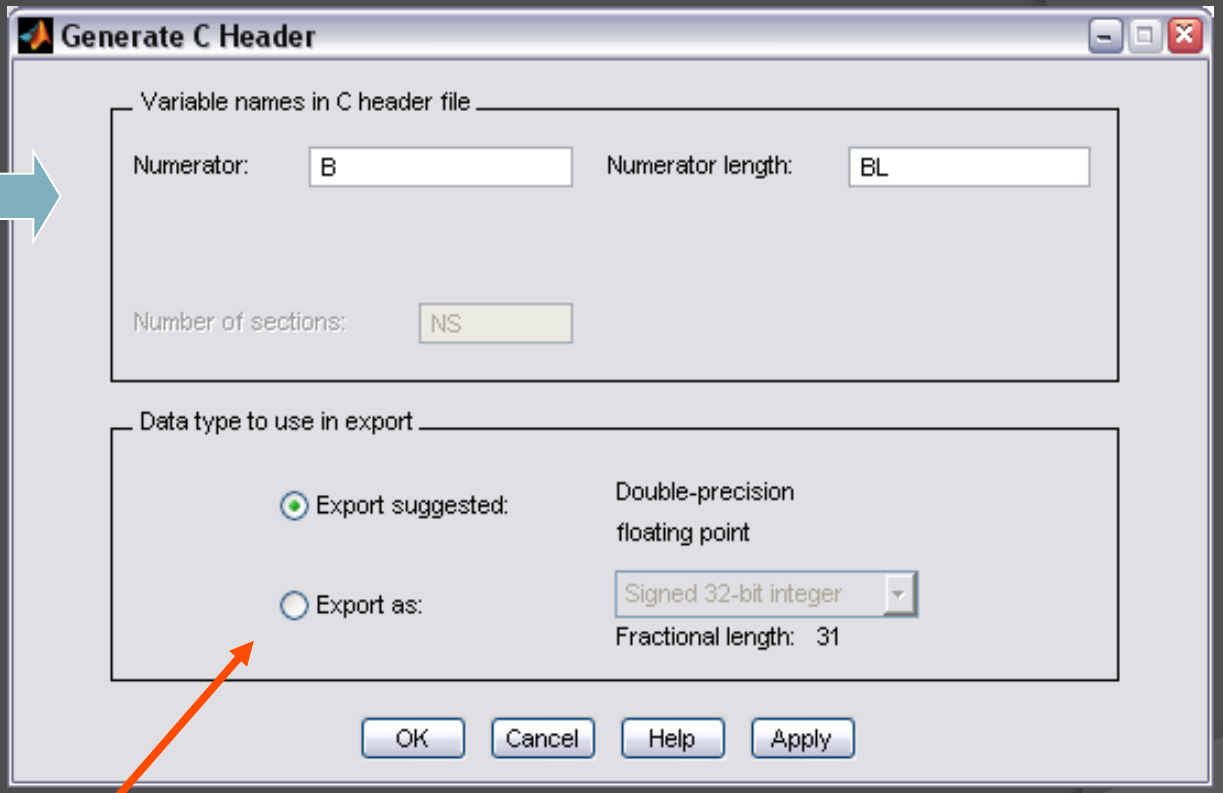
(picture from Matlab's help system)

Compute FIR Filter Coefficients



Make Coefficient File For CCS





Here you can change the coefficient data type to match your desired coefficient quantization.

Main Datatypes for FIR/IIR Filtering

- Signed integer:
  - (8 bit) signed char: -128 to +127
  - (16 bit) short: -32768 to +32767
  - (32 bit) int: -215E6 to 215E6
- Floating point:
  - (32 bit) float: -3.4E38 to +3.4E38 with numbers as small as 1.175E-38
  - (64 bit) double: -1.7E308 to +1.7E308 with numbers as small as 2.2E-308

Example DP-FP Coefficient File

```
/*
 * Filter Coefficients (C Source) generated by the Filter Design and Analysis Tool
 *
 * Generated by MATLAB(R) 7.0 and the
 *
 * Generated on: 19-Aug-2005 13:04:09
 *
 */

/*
 * Discrete-Time FIR Filter (real)
 * -------------------------------
 * Filter Structure  : Direct-Form FIR
 * Filter Order      : 8
 * Stable            : Yes
 * Linear Phase      : Yes (Type 1)
 */

/* General type conversion for MATLAB generated C-code  */
#include "tmwtypes.h"
/*
 * Expected path to tmwtypes.h
 * C:\MATLAB7\extern\include\tmwtypes.h
 */
const int BL = 9;
const real64_T B[9] = {
    0.02588139692752,  0.08678803067191,   0.1518399865268,   0.2017873498839,
     0.2205226777929,   0.2017873498839,   0.1518399865268,  0.08678803067191,
    0.02588139692752
};
```

Note this new header file needed for CCS to understand Matlab's strange data types.

Add this header file to your project (in the Matlab directory tree) or edit the datatypes.

FIR Filter Coefficient Quantization Considerations

- Key choice: floating point vs. fixed point
- Advantages of floating point math:
  - Less quantization error (more precision)
  - Don't have to worry about overflow
  - Don't have to worry about keeping track of scaling factors
  - Much easier to code
- Disadvantages of floating point math:
  - Executes slower than fixed point
  - Requires you to use a floating-point DSP ($$$, power, heat,…)
- C code allows you to "cast" variables into any datatype

Casting Variables in C

```
short a,b,c;                    // 16-bit signed integers
double x,y,z;                   // double-precision float

x = 456.78;
a = (short) x;

a = -4321;
x = (double) a;

x = 33333;
a = (short) x;                  // What happens here?
```

Note: Type casting takes precedence over most math operators.

Write Code to Realize FIR Filter

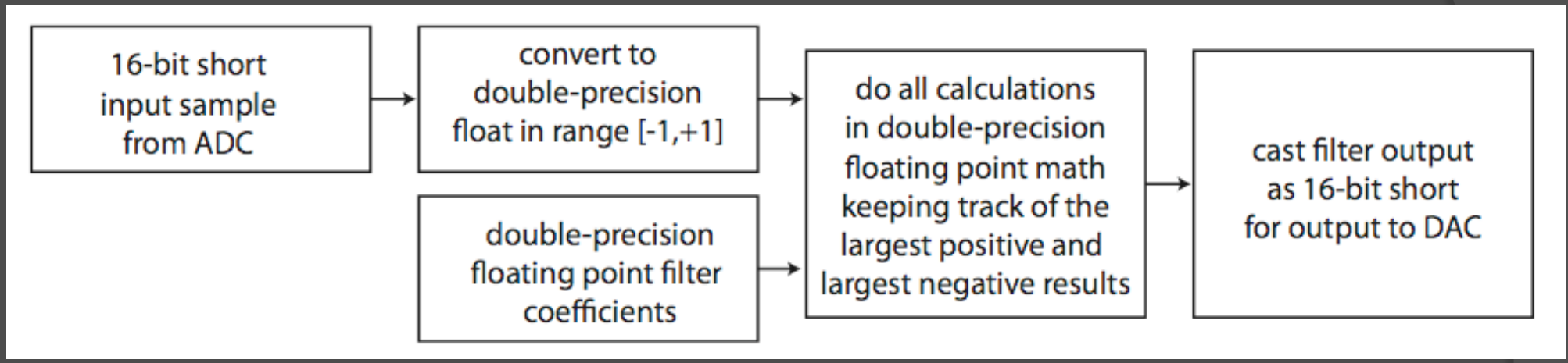
- Direct form I implies direct realization of the convolution equation (multiply and accumulate)

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n-m]$$

- Some practical considerations:
  - Allocate buffer of length M for filter coefficients.
  - Allocate buffer of length M for input samples.
  - Move input buffer pointer as new data comes in or move data?

WORCESTER POLYTECHNIC INSTITUTE 1865 WPI

Double-Precision Floating Point Filter Realization



- Since everything is DP-FP, you don't need to worry about overflow (except at the output)
- Keeping track of the largest positive and largest negative intermediate results is optional, but will help with:
  - Detecting overflow in the output (short)
  - Designing a fixed-point implementation with proper scaling factors that avoids overflow (Lab 3)

Verifying your real-time filter works correctly

- Method 1: Sinusoids (easy but labor intensive)
  - Make a table with columns for f, a_{in}, and a_{out}
  - Generate input sinusoid at frequency f with amplitude a_{in}.
  - LTI filter output will also be at frequency f but with amplitude a_{out}.
  - Magnitude response of the filter is $20\log_{10}(a_{out}/a_{in})$
  - Compare actual magnitude response to the predicted response from Matlab
- Method 2: White noise (more complicated but less work)
  - Generate at least 10 seconds of a white noise input signal (matlab command rand or randn)
  - Record your digital filter output to a .wav file
  - Use Matlab commands wavread and pwelch to estimate "power spectral density" of the digital filter output