# SIGNAL PROCESSING WITH MATLAB

# *What is signal Processing?*

- The scope of signal processing has grown so broad as to obviate a perfect and precise definition of what is entailed in it[1].
- Traditionally, signal processing includes the materials thought in DSP courses but now signal processing has greater reach because of its influence on related disciplines such as controls, communications theory and also digital communication.
- Thus, signal processing can be defined as that area of applied mathematics that deals with *operations on or analysis of signals,* in either discrete or continuous time, to perform useful operations on those signals[1].

# ***What is Signal Processing Toolbox?***

The Signal Processing Toolbox is a collection of tools built on the MATLAB® numeric computing environment. The toolbox supports a wide range of signal processing operations, from waveform generation to filter design and implementation, parametric modeling, and spectral analysis. The toolbox provides **two** categories of tools.

# ***Command line functions*** in the following categories:

- Analog and digital filter analysis
- Digital filter implementation
- FIR and IIR digital filter design
- Analog filter design
- Filter discretization
- Spectral Windows Transforms
- Statistical signal processing and spectral analysis
- Parametric modeling
- Linear Prediction
- Waveform generation

# **A suite of *interactive graphical user interfaces (GUI)*for**

- Filter design and analysis
- Window design and analysis
- Signal plotting and analysis
- Spectral analysis
- Filtering signals

# Signal Processing Basics

## *Common Sequences*

- Since MATLAB is a programming language, an endless variety of different signals is possible. Here are some statements that generate several commonly used sequences, including the unit impulse, unit step, and unit ramp functions:

```
t = (0:0.01:1);
y = ones(101);              % step
y = [1; zeros(100,1)];      % impulse
y = t  ;                    % ramp
y = t.^2;                   % exponential
y = square(2*pi*4*t);       % generates a square wave every 0.25secs.
```

# *Waveform generation*

- y = sin(2*pi*50*t) + 2*sin(2*pi*120*t); %two sinusoids,
  %one at 50 Hz
  %and one at
  %120Hz with
  %twice the amplitude

- plot(t,y) %plot y versus time
- plot(t(1:50),y(1:50)) %display only the first
  %50 points(zoom!)

# Filter Implementation and Analysis

## *Convolution and Filtering*

The mathematical foundation of filtering is convolution. The MATLAB conv function performs standard one-dimensional convolution, convolving one vector with another

$$y(k) = h(k) * x(k) = \sum_{l=-\infty}^{\infty} h(k-l)x(l)$$

A digital filter's output $y(k)$ is related to its input $x(k)$ by convolution with its impulse response $h(k)$.

# *Cont*.

- x = [1 2 1];
- h = [1 1 1];
- y = conv(h,x);
- stem(y)

# *Filters and Transfer Functions*

In general, the *z*-transform *Y*(*z*) of a digital filter's output *y*(*n*) is related to the *z*-transform *X*(*z*) of the input by

$$Y(z) = H(z)X(z) = \underbrace{\frac{b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}}{a(1) + a(2)z^{-1} + \cdots + a(m+1)z^{-m}}}_{H(z)} X(z)$$

where *H*(*z*) is the filter's *transfer function*. Here, the constants *b*(*i*) and *a*(*i*) are the filter coefficients and the order of the filter is the maximum of *n* and *m*.

# *Cont.*

- For example let;

$$H(z) = \frac{1}{1 - 0.9z^{-1}}$$

- step = ones(50); %input data : step function
- b = 1; % Numerator
- a = [1 -0.9]; % Denominator

where the vectors b and a represent the coefficients of a filter in transfer function form. To apply this filter to your data, use

- y = filter(b,a,step);
- stem(y)

- fvtool(b,a) %GUI.Don't have to define input (if input is %step/impulse function)%

# Filter Design and Implementation

- Filter design is the process of creating the filter coefficients to meet specific filtering requirements. Filter implementation involves choosing and applying a particular filter structure to those coefficients.

- Only after both design and implementation have been performed can data be filtered.

# *Filter Coefficients and Filter Names*

In general, the *z*-transform *Y*(*z*) of a digital filter's output *y*(*n*) is related to the *z*-transform *X*(*z*) of the input by

$$Y(z) = H(z)X(z) = \frac{b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}}{a(1) + a(2)z^{-1} + \cdots + a(m+1)z^{-m}}X(z)$$

- Many standard names for filters reflect the number of a and b coefficients present:
  - When **n = 0** (that is, b is a scalar), the filter is an Infinite Impulse Response (**IIR**), all-pole, recursive, or autoregressive (AR) filter.
  - When **m = 0** (that is, a is a scalar), the filter is a Finite Impulse Response **(FIR),** all-zero, nonrecursive, or moving-average (MA) filter.
  - If both n and m are greater than zero, the filter is an IIR, pole-zero, recursive, or autoregressive moving-average (ARMA) filter.

# *IIR Filter Design*

- The primary advantage of IIR filters over FIR filters is that they typically meet a given set of specifications with a much lower filter order than a corresponding FIR filter.

# *Complete Classical IIR Filter Design*

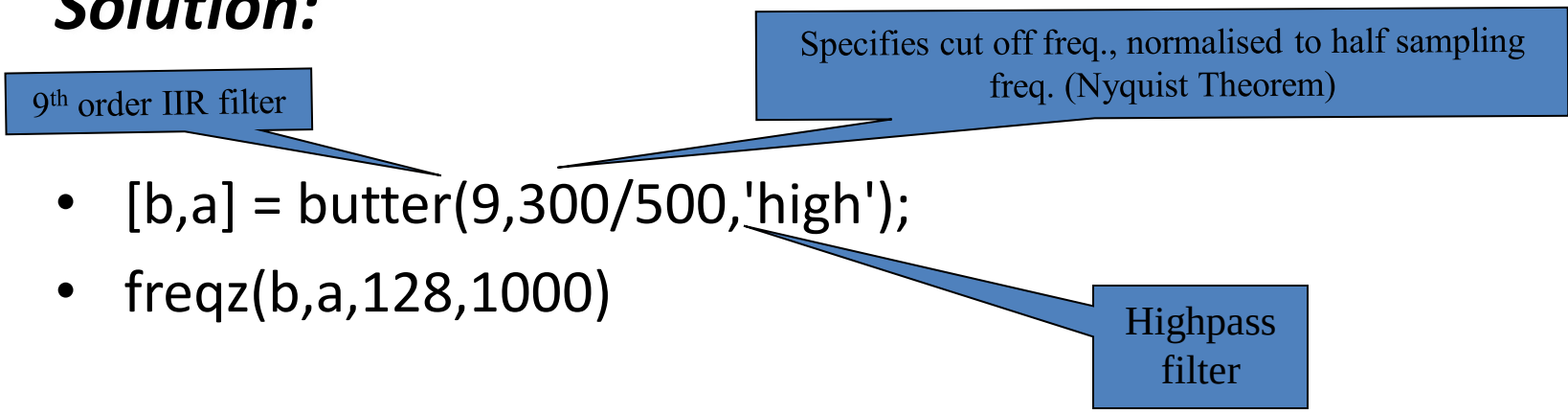
| **Filter Type** | **Design Function** |
|---|---|
| Bessel (analog only) | [b,a] = besself(n,Wn,*options*)<br>[z,p,k] = besself(n,Wn,*options*)<br>[A,B,C,D] = besself(n,Wn,*options*) |
| Butterworth | [b,a] = butter(n,Wn,*options*)<br>[z,p,k] = butter(n,Wn,*options*)<br>[A,B,C,D] = butter(n,Wn,*options*) |
| Chebyshev Type I | [b,a] = cheby1(n,Rp,Wn,*options*)<br>[z,p,k] = cheby1(n,Rp,Wn,*options*)<br>[A,B,C,D] = cheby1(n,Rp,Wn,*options*) |
| Chebyshev Type II | [b,a] = cheby2(n,Rs,Wn,*options*)<br>[z,p,k] = cheby2(n,Rs,Wn,*options*)<br>[A,B,C,D] = cheby2(n,Rs,Wn,*options*) |
| Elliptic | [b,a] = ellip(n,Rp,Rs,Wn,*options*)<br>[z,p,k] = ellip(n,Rp,Rs,Wn,*options*)<br>[A,B,C,D] = ellip(n,Rp,Rs,Wn,*options*) |

# *Cont*

***Example 1:***

- For data sampled at 1000 Hz, design a 9th-order highpass Butterworth IIR filter with cutoff frequency of 300 Hz,

***Solution:***




- [b,a] = butter(9,300/500,'high');
- freqz(b,a,128,1000)

# *Cont.*

*Example 2:*

For data sampled at 1000 Hz, design a 9th-order lowpass Chebyshev Type I filter with 0.5 dB of ripple in the passband and a cutoff frequency of 300 Hz, which corresponds to a normalized value of 0.6:

*Solution*

```
[b,a] = cheby1(9,0.5,300/500);
freqz(b,a,512,1000)%The frequency response of the filter
```

# *FIR Filter Design*

Digital filters with finite-duration impulse response (all-zero, or FIR filters) have both advantages and disadvantages compared to infinite-duration impulse response (IIR) filters.

FIR filters have the following primary *advantages*:

- They can have exactly linear phase.
- They are always stable.
- The design methods are generally linear.
- They can be realized efficiently in hardware.
- The filter startup transients have finite duration.

The *primary disadvantage* of FIR filters is that they often require a much higher filter order than IIR filters to achieve a given level of performance. Correspondingly, the delay of these filters is often much greater than for an equal performance IIR filter.

| Method | Description | Functions |
| --- | --- | --- |
| Windowing | Apply window to truncated inverse Fourier transform of desired "brick wall" filter | fir1, fir2, kaiserord |
| Multiband with Transition Bands | Equiripple or least squares approach over sub-bands of the frequency range | firls, remez, remezord |
| Constrained Least Squares | Minimize squared integral error over entire frequency range subject to maximum error constraints | fircls, fircls1 |
| Arbitrary Response | Arbitrary responses, including nonlinear phase and complex filters | cremez |
| Raised Cosine | Lowpass response with smooth, sinusoidal transition | firrcos |

# *Cont.*

- *Example 1*

  Design a 48th-order FIR bandpass filter with passband 0.35 0.65:

  $\leq \omega \leq$

- ***Solution***

  - b = fir1(48,[0.35 0.65]);
  - freqz(b,1,512)

# *Cont.*

- ***Example 2:***

  Design a lowpass filter with the following specifications using the optimal design method :

```
rp = 0.01;          % Passband ripple
rs = 0.1;           % Stopband ripple
fs = 8000;          % Sampling frequency
f = [1500 2000];    % Cutoff frequencies
a = [1 0];          % Desired amplitudes
```

# *Cont.*

a vector of maximum deviations or ripples allowable for each band.

- ***Solution***

%[n,fo,ao,w] = remezord(f,a,dev,fs); %

dev=[0.01 0.1]

[n,fo,ao,w]=remezord([1500 2000],[1 0],dev,8000); % approximate order, normalized frequency band edges, frequency band amplitudes, and weights that meet input specifications f, a, and dev.%

b=remez(n,fo,ao,w); %use n, fo, ao and w to design the filter b which approximately meets the specifications given by remezord input parameters f, a, and dev.%

freqz(b,1,1024,8000);

title('Lowpass Filter Designed to Specifications');

# *Filter Implementation*

After the filter design process has generated the filter coefficient vectors, b and a, two functions are available in the Signal Processing Toolbox for implementing the filter:

- filter-:for b and a coefficient input, implements a direct-form II transposed structure and filters the data. For dfilt input, filter uses the structure specified with dfilt and filters the data.

- Dfilt-:let us specify the filter structure and creates a digital filter object.

# *Cont.*

- ***Example using filter***

- t = (0:0.001:1); %fs=1/0.001Hz
- x = sin(2*pi*50*t) + 2*sin(2*pi*120*t); %define input
- plot(t,x) %plot input
- [b,a] = butter(9,100/500,'high'); %9$^{th}$ order,high pass filter
  %with cutoff freq. 100Hz
- c=filter(b,a,x);
- figure(2)
- plot(t,c)

# *Cont.*

- The complete process of designing, implementing, and applying a filter using a dfilt object is described below:

    1. Generate the filter coefficients using any IIR or FIR filter design function.
    2. Create the filter object from the filter coefficients and the specified filter structure using dfilt.
    3. Apply the dfilt filter object to the data, x using filter.

# *Cont.*

- ***Example using dfilt***

- t = (0:0.001:1);
- x = sin(2*pi*50*t) + 2*sin(2*pi*120*t); %define input
- plot(t,x)
- [b,a] = butter(9,100/500,'high'); %design filter
- Hd = dfilt.df2t(b,a); %Implement direct-form II transposed
- c=filter(Hd,x);
- figure(2)
- plot(t,c)

returns a discrete-time filter object, Hd, of type *df2t(direct-form II transposed*

# Statistical Signal Processing

## *Correlation and Covariance*

The functions xcorr and xcov estimate the cross-correlation and cross-covariance sequences of random processes.

The cross-correlation sequence is a statistical quantity defined as

$$R_{xy}(m) = E\{x_{n+m}y^*_n\} = E\{x_n y^*_{n-m}\}$$

where $x_n$ and $y_n$ are stationary random processes, $-\infty < n < \infty$ , and $E\{\cdot\}$ is the expected value operator which measures the similarity between the two waveforms.The covariance sequence is the mean-removed cross-correlation sequence

$$C_{xy}(m) = E\{(x_{n+m} - \mu_x)(y_n - \mu_y)^*\}$$

# *Cont.*

or, in terms of the cross-correlation,

$$C_{xy}(m) = R_{xy}(m) - \mu_x \mu^*{}_y$$

# *Cont.*

- ***Example on xcorr***

  - x = [1 1 1 1 1]';
  - y = x;
  - xyc = xcorr(x,y)
  - stem(xyc)

- ***Example on xcov***

```
ww = randn(1000,1);  % Generate uniform noise with mean = 1/2.%
[cov_ww,lags] = xcov(ww,10,'coeff');
stem(lags,cov_ww)
```

# *Fast Fourier Transform (FFT)*

- The discrete Fourier transform, or DFT, is the primary tool of digital signal processing. The foundation of the Signal Processing Toolbox is the fast Fourier transform (FFT), a method for computing the DFT with reduced execution time. Many of the toolbox functions (including *z*-domain frequency response, spectrum and cepstrum analysis, and some filter design and implementation functions) incorporate the FFT.

# *Cont.*

```
t = (0:0.001:1);                                %0.001 is sampling
x = sin(2*pi*50*t) + 2*sin(2*pi*120*t);
y = fft(x);                                     %Compute DFT of x
m = abs(y);                                     %magnitude
f = (0:length(y)/2-1)*1000/length(y);   %Frequency vector
plot(f,m(1:1:(length(m)-1)/2))          %before filter
grid
[b,a] = butter(9,100/500,'high');       %design filter
c=filter(b,a,x);                        %implement filter
figure(2)
y = fft(c);                                     %Compute DFT of c(filtered x)
m = abs(y);                                     % Magnitude
f = (0:length(y)/2-1)*1000/length(y);  %Frequency vector
plot(f,m(1:1:(length(m)-1)/2))          %after filter
Grid
```

# FDATool: A Filter Design and Analysis GUI

- The Filter Design and Analysis Tool (FDATool) is a powerful user interface for designing and analyzing filters. FDATool enables you to quickly design digital FIR or IIR filters by setting filter performance specifications, by importing filters from your MATLAB workspace, or by directly specifying filter coefficients.

- FDATool also provides tools for analyzing filters, such as magnitude and phase response plots and pole-zero plots. You can use FDATool as a convenient alternative to the command line filter design functions.

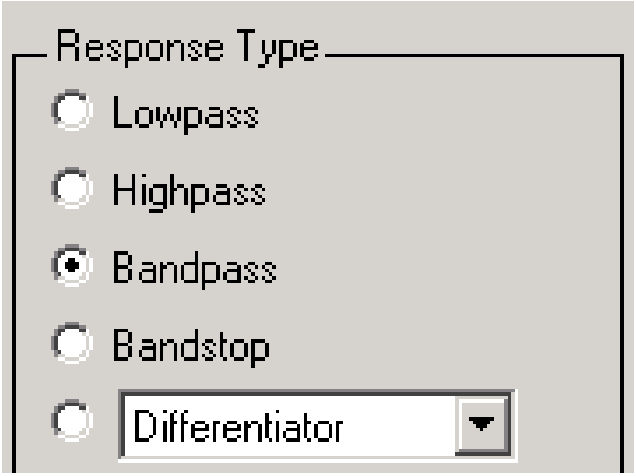
# *Opening FDATool*

- To open the Filter Design and Analysis Tool, type

  » fdatool

## *Choosing a Response Type*

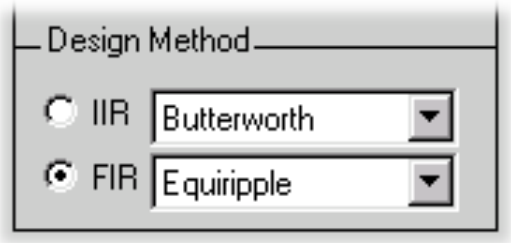
You can choose from several response types:



# *Choosing a Filter Design Method*

- You can use the default filter design method for the response type that you've selected, or you can select a filter design method from the available FIR and IIR methods listed in the GUI.

- To select the Remez algorithm to compute FIR filter coefficients, select the **FIR** radio button and choose Equiripple from the list of methods.

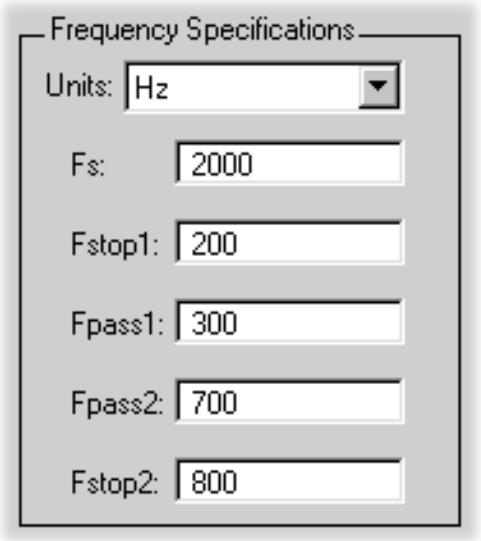


# *Setting the Filter Design Specifications*

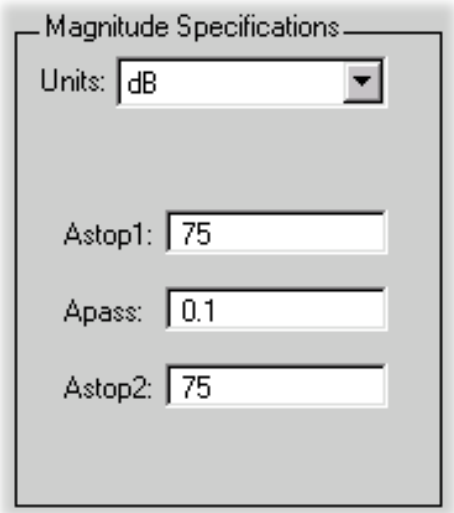
- The filter design specifications that you can set vary according to response type and design method. For example, to design a bandpass filter, you can enter

  - » Filter order
  - » Options
  - » Bandpass Filter Frequency Specifications
  - » Bandpass Filter Magnitude Specifications

- ***Bandpass Filter Frequency Specifications***
  - For this example:



- ***Bandpass Filter Magnitude Specifications***
  - For this example




- ***Computing the Filter Coefficients***

  Now that you've specified the filter design, click the Design Filter button to compute the filter coefficients.

# References


1. Tood K. Moon, Wynn C. Stirling “Mathematical Methods and Algorithms for Signal Processing”,Prentice Hall,2000.
2. Samuel D. Stearns, Ruth A. David “Signal Processing Algorithms In Matlab”, Prentice Hall,1996.
3. http://www.mathworks.com.