Lesson 7
Signal Processing toolbox
DSP blockset

Instructor:
ir drs E.J Boks
Elecrical Engineering
Embedded Systems Engineering
phone:(026) 3658173
Room: D2.03
e-mail:ewout.boks@han.nl

What is DSP ?

- *Digital Signal Processing* (*DSP*) is concerned with the digital representation of signals and the use of digital processing to analyse, modifiy or extract information from signals.
- Most signals in nature are analogue in form, meaning that they vary continously with time. Signals that are used in DSP are derived from analogue signals. They have been sampled at regular intervals and as such they are of a discrete nature.

Why DSP ?

DSP has a number of advantages over analogue signal processing:

- Guaranteed accuracy
- Perfect reproducibility
- No drift with time or temperature
- Miniaturisation of implementation based on VLSI
- Greater flexibility
- Superior performance

DSP bottlenecks

DSP doesn't come without a price. The following issues are of concern when using DSP:

- Speed and cost
- Design and time
- System resolution vs cost

Two tools

- Matlab/Simulink offers two different toolsets for DSP related engineering:
  - The Signal Processing toolbox (Matlab)
  - The DSP blockset (Simulink)

Signal Processing Toolbox

- The toolbox supports a wide range of signal processing operations, from waveform generation to filter design and implementation, parametric modeling, and spectral analysis.

Signal Toolbox contents

- The toolbox provides two categories of tools:
  - Command line tools
  - GUI based tools

Command line tools

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

Interactive GUI based tools

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

Signals in Matlab

- Please recall that Matlab always works with discrete signals (even when representing analogue signals).
- For MATLAB being a scripting tool, 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:
- Common Sequences: Unit Impulse, Unit Step, and Unit Ramp
- t = (0:0.001:1)';
- y = [1; zeros(99,1)]; % impulse
- y = ones(100,1); % step (filter assumes 0 initial cond.)
- y = t; % ramp
- y = t.^2;
- y = square(4*t);

Signals in Matlab

- Signals may be derived from functions. Special signal functions are:
  - sawtooth generates a sawtooth wave with peaks at ±1 and a period of 2π. An optional width parameter specifies a fractional multiple of at which the signal's maximum occurs.
  - square generates a square wave with a period of 2π. An optional parameter specifies duty cycle, the percent of the period for which the signal is positive.
  - The sinc function computes the mathematical sinc function for an input vector or matrix x. The sinc function is the continuous inverse Fourier transform of the rectangular pulse of width and height 1.

Signals in Matlab

- Outside data can be read into matlab for analysis. Depending on your data format, you can do this in the following ways:
  - Load data from an ASCII file or MAT-file with the MATLAB load command.
  - Read the data into MATLAB with a low-level file I/O function, such as fopen, fread, and fscanf.

Basic signal processing functions

- Two basic signal processing functions exist in the Matlab basic package, namely:
  - filter : This function filters a data sequence using a digital filter which works for both real and complex inputs. The filter is a direct form II transposed implementation of the standard difference equation.
  - fft : This function performs the discrete Fourier Transform.

Data input

- In Signal theory, systems usually are characterised by their transfer functions in either the s domain or z domain.
  - S domain : $$H(s)=\frac{b_1 s^n + b_2 s^{n-1} + ... + b_{n+1}}{s^m + a_1 s^{m-1} + ... + a_{m-1}}$$
  - Z domain : $$H(z)=\frac{b_1 + b_2 z^{-1} + ... + b_{n+1} z^{-n}}{1 + a_1 z^{-1} + ... + a_m z^{-m}}$$
- Coefficients a and b are entered in rows where their index number in the row denotes their index in the transfer function as listed above.

Transfer functions

- For example the filter function prototype is listed as:

```
y = filter(b,a,X);
```

- b and a are the coefficients of the filter as given in the previous slide and X is the input signal.

Visualisation of data

In order to visualize data, several functions exist. Consider this example:

The impulse response of a digital filter is the output arising from the unit impulse input sequence defined as equal to one when n=0 and equal to zero when $n \neq 0$.

In MATLAB, you can generate an impulse sequence in a number of ways; one straightforward way is

```
imp = [1; zeros(49,1)];
```

The impulse response of the simple filter b = 1 and a = [1 -0.9] is

```
h = filter(b,a,imp);
```

A simple way to display the impulse response is with the Filter Visualization Tool (fvtool):

```
fvtool(b,a)
```

Visualisation of frequency response

- To visualise a frequency response of a system, the following two tools come in handy:
  - [h,w] = freqz(b,a,l) returns the frequency response vector **h** and the corresponding angular frequency vector **w** for the *digital* filter whose transfer function is determined by the (real or complex) numerator and denominator polynomials represented in the vectors b and a, respectively. The vectors h and w are both of length l. The angular frequency vector w has values ranging from 0 to radians per sample. When you don't specify the integer l, or you specify it as the empty vector [], the frequency response is calculated using the default value of 512 samples.
  - h = freqs(b,a,w) returns the complex frequency response of the *analog* filter specified by coefficient vectors b and a. freqs evaluates the frequency response along the imaginary axis in the complex plane at the angular frequencies in rad/sec specified in real vector w, which must contain more than one frequency.

Filter design

- Filter design is a major topic in (Digital) Signal Processing.
- Two classes of filters: Analog and digital filters.
- Two subclasses exist: Infinite Impulse Response (IIR) and Finite Impulse Response (FIR, exists solely in the discrete domain).

Analog filter design

- Analog filter design is supported with the following functions:
  - Bessel
    - [b,a] = besself(n,Wn,options)
    - [z,p,k] = besself(n,Wn,options)
    - [A,B,C,D] = besself(n,Wn,options)
  - Butterworth
    - [b,a] = butter(n,Wn,options)
    - [z,p,k] = butter(n,Wn,options)
    - [A,B,C,D] = butter(n,Wn,options)

Analog filter design

- Chebyshev Type I
  - [b,a] = cheby1(n,Rp,Wn,options)
  - [z,p,k] = cheby1(n,Rp,Wn,options)
  - [A,B,C,D] = cheby1(n,Rp,Wn,options)
- Chebyshev Type II
  - [b,a] = cheby2(n,Rs,Wn,options)
  - [z,p,k] = cheby2(n,Rs,Wn,options)
  - [A,B,C,D] = cheby2(n,Rs,Wn,options)

Analog filter design

- Elliptic
  - [b,a] = ellip(n,Rp,Rs,Wn,options)
  - [z,p,k] = ellip(n,Rp,Rs,Wn,options)
  - [A,B,C,D] = ellip(n,Rp,Rs,Wn,options)

Filter design requirements

- Filters are usually specified with operational performance parameters. In order to calculate the minimally required filter size, use the following tools:
  - Butterworth
    - [n,Wn] = buttord(Wp,Ws,Rp,Rs)
  - Chebyshev Type I
    - [n,Wn] = cheb1ord(Wp, Ws, Rp, Rs)
  - Chebyshev Type II
    - [n,Wn] = cheb2ord(Wp, Ws, Rp, Rs)
  - Elliptic
    - [n,Wn] = ellipord(Wp, Ws, Rp, Rs)

Requirements example

These are useful in conjunction with the filter design functions. Suppose you want a bandpass filter with a passband from 1000 to 2000 Hz, stopbands starting 500 Hz away on either side, a 10 kHz sampling frequency, at most 1 dB of passband ripple, and at least 60 dB of stopband attenuation. You can meet these specifications by using the butter function as follows:

```
[n,Wn] = buttord([1000 2000]/5000,[500 2500]/5000,1,60)

n =

   12

Wn =

   0.1951    0.4080

[b,a] = butter(n,Wn);
```

Analogue filter example

Requirements: design an analogue lowpass filter with :

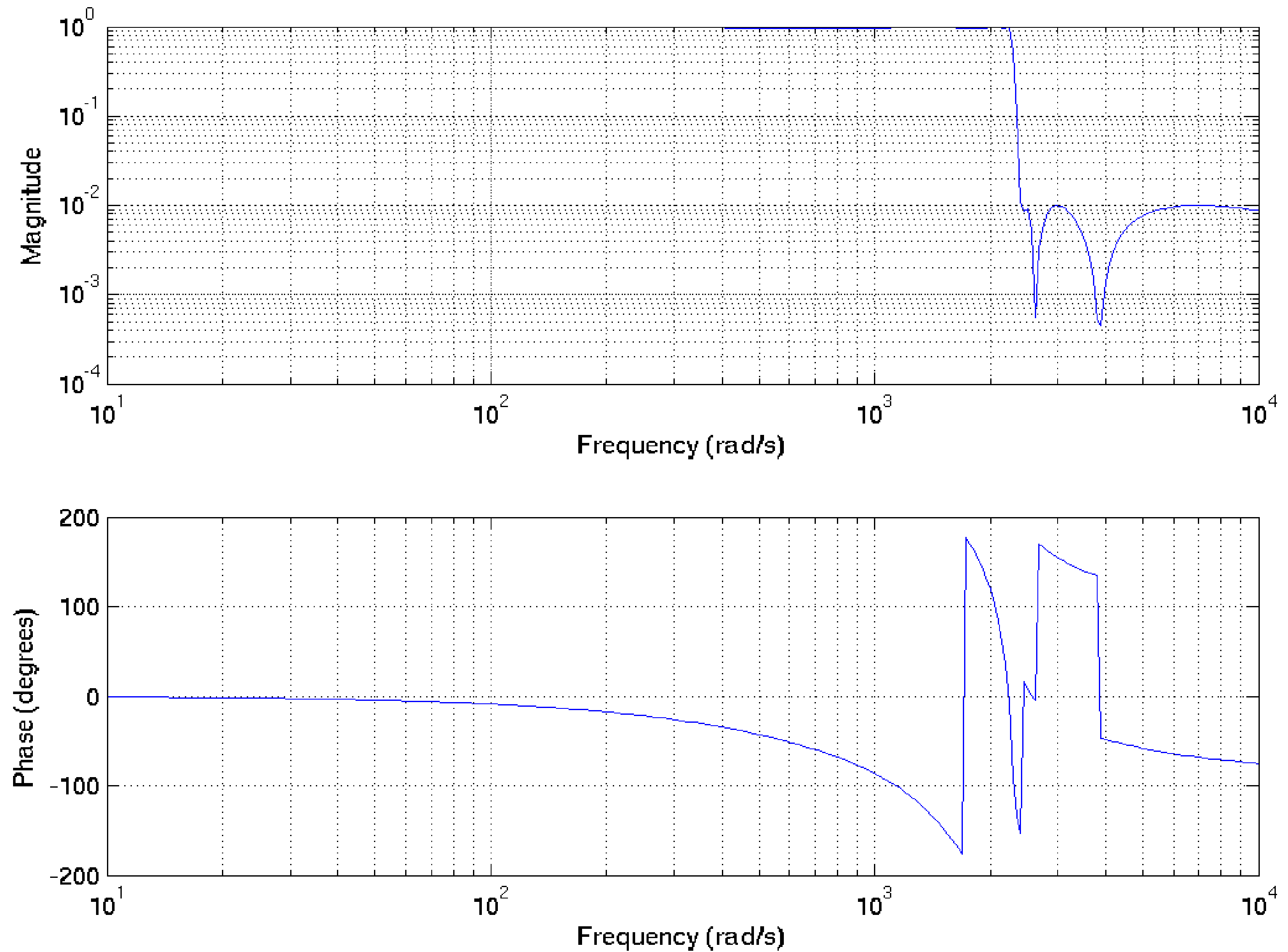
- Stopband attenuation = 40 dB
- Sharpest roll-off possible
- Ripple in passband acceptable to 0.5 dB
- Cutoff frequency is 360 Hz

Analog filter example

```
% Lesson 7
% MSCS course, Hogeschool van Arnhem en Nijmegen
% example 1
% lowpass filter

angularfreq = 2*pi;
cutoff = 360;
passripple = 0.5;
stopatt = 40;
% Steepest roll-off possible --> we select an elliptic filter
% determine the order
[n Wn] = ellipord(cutoff*angularfreq,cutoff*1.1*angularfreq, passripple, ...
  stopatt, 's');
% design the filter
[b a] = ellip(n,passripple,stopatt,Wn,'s');
% show the response
freqs(b,a);
```

Analog filter example



Exercise1

In a medical application, heart-beat audio data is sampled.

During the sampling session, it is unavoidable that certain baseline wander and artefacts are introduced into the signal due to patient body movement.

Design an analogue filter that meets the following requirements to remove these unwanted signal additions:

- Passband: 10-45 Hz
- Stopband 0-5 Hz and 50 – inf Hz
- Stopband ripple: equal to 5 dB
- Stopband attenuation: larger than 20 dB
- Use a Chebychev type II filter

Digital filter design

- IIR class of filters :
  - Analog prototyping: use the analog design functions, but specify a sampling frequency with the options.
  - Direct design: Design digital filter directly in the discrete time-domain by approximating a piecewise linear magnitude response.
  - Generalized Butterworth design: Design lowpass Butterworth filters with more zeros than poles.
  - Parametric modelling: find a digital filter that approximates a prescribed time or frequency domain response.

Digital filter design

- FIR class of filters:
  - 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

Digital filter example

Filter requirements:

- Sampling frequency 1 kHz
- High pass filter, cutoff at 200 Hz
- No phase distortions in passband
- Max order = 20

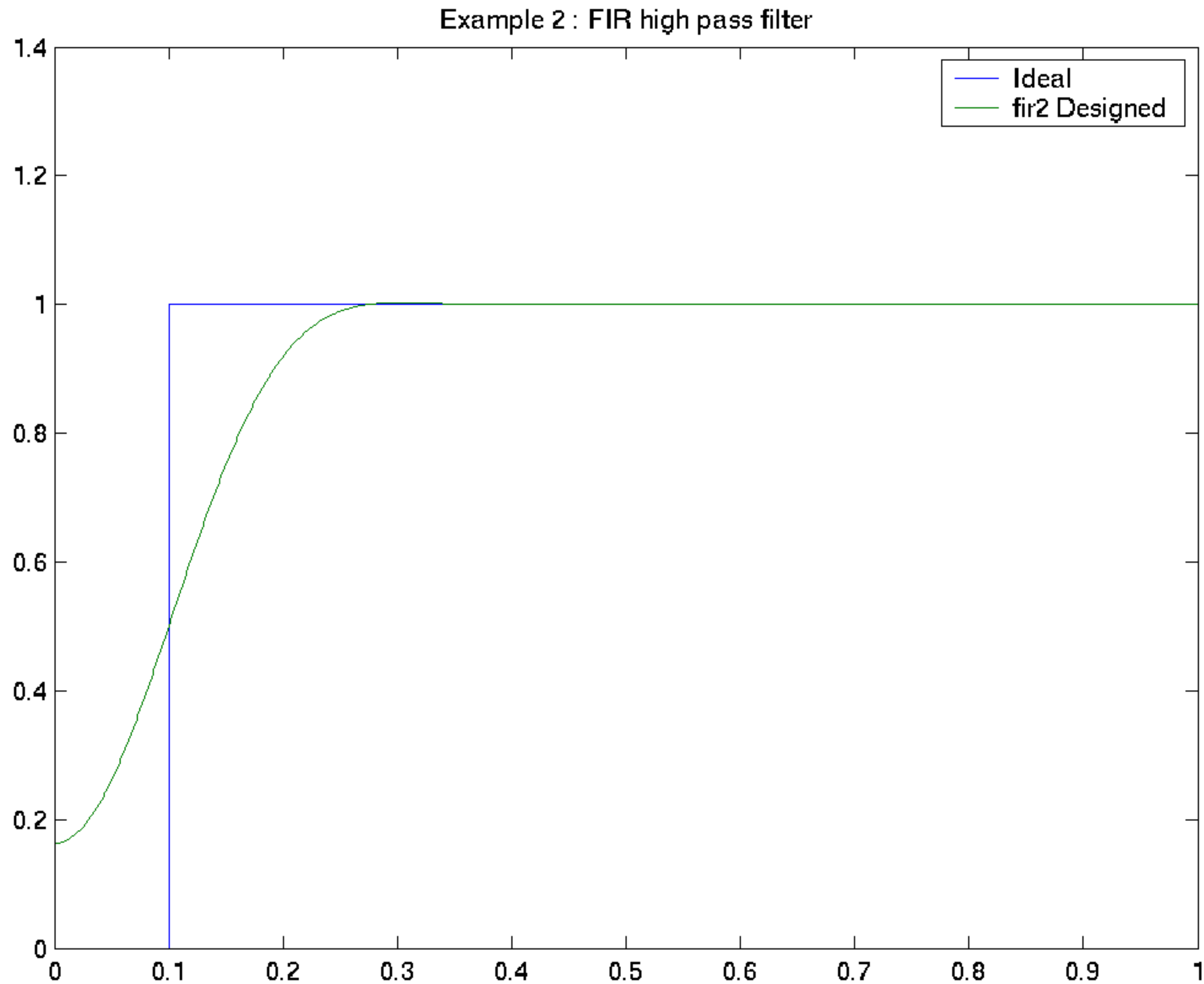
Digital filter example

```
% Lesson 7
% MSCS course, Hogeschool van Arnhem en Nijmegen
% example 2
% digital highpass filter
samplefreq = 1E3;
cutoff = 200;
order = 20;
% No phase distortions --> FIR filter
% the desired frequency window
```

Digital filter example (continued)

```
f = [0 cutoff/(2*samplefreq) cutoff/(2*samplefreq) 1];
m = [0 0 1 1];
% design the filter
b = fir2(order,f,m);
% show the response
[h w] = freqz(b,1,samplefreq);
plot(f,m,w/pi,abs(h))
legend('Ideal','fir2 Designed')
title('Example 2 : FIR high pass filter')
```

Digital filter example



Signal processing analysis tool

- A tool exists to quickly evaluate and design signal processing data or filters. This is the Signal & Analysis Processing (SPA) tool.
- To start the SPA tool, type sptool at the cmd prompt. Import the a and b vectors from example 1, you can evaluate the filter's performance more interactively.

Filter design tool

- Combining most filter design tools and the sptool into one design tool: the Filter Design and Analysis (FDA) tool.
- The tools is an excellent and very powerful, easy to use GUI-based tool that lets the designer quickly specify, design and evaluate any type of filter.
- To start the FDA tool, type fdatool on the cmd line.

Statistical filter design

- The Signal Processing Toolbox provides tools for estimating important functions of random signals.
- In particular, there are tools to estimate correlation and covariance sequences and spectral density functions of discrete signals.

Correlation and covariance calculation

- The functions xcorr and xcov estimate the cross-correlation and cross-covariance sequences of random processes. They also handle autocorrelation and autocovariance as special cases.

Example:

```
x = [1 1 1 1 1]';

y = x;

xyc = xcorr(x,y)

xyc =

   1.0000 2.0000 3.0000 4.000 5.0000 4.0000 3.0000 2.0000 1.0000
```

Signal spectral analysis

- The goal of spectral estimation is to describe the distribution (over frequency) of the power contained in a signal, based on a finite set of data. Estimation of power spectra is useful in a variety of applications, including the detection of signals buried in wide-band noise.

Spectral analysis tools

- Periodogram
  - Power spectral density estimate: periodogram
- Welch
  - Averaged periodograms of overlapped, windowed signal sections: pwelch, csd, tfe, cohere
- Multitaper
  - Spectral estimate from combination of multiple orthogonal windows (or "tapers"): pmtm
- Yule-Walker AR
  - Autoregressive (AR) spectral estimate of a time-series from its estimated autocorrelation function: pyulear
- Burg
  - Autoregressive (AR) spectral estimation of a time-series by minimization of linear prediction errors: pburg

Spectral analysis tools

- Covariance
  - Autoregressive (AR) spectral estimation of a time-series by minimization of the forward prediction errors: pcov
- Modified Covariance
  - Autoregressive (AR) spectral estimation of a time-series by minimization of the forward and backward prediction errors: pmcov
- MUSIC
  - Multiple signal classification: pmusic
- Eigenvector
  - Pseudospectrum estimate: peig

DSP blockset overview

The Matlab DSP Blockset is a collection of block libraries for use with the Simulink dynamic system simulation environment.

The DSP Blockset libraries are designed specifically for digital signal processing (DSP) applications, and include key operations such as

- *Classical filters*
- *Multirate filters*
- *adaptive filtering*
- *matrix manipulation*
- *linear algebra*
- *Statistics*
- *time-frequency transforms*

DSP Blockset overview II

The DSP blockset therefore has the following features:

- Frame-Based Operations
- Matrix Support
- Adaptive and Multirate Filtering
- Statistical Operations
- Linear Algebra
- Parametric Estimation
- Real-Time Code Generation (optional)

DSP Blockset Features

Frame-Based Operations:

- Most real-time DSP systems optimize throughput rates by processing data in "batch" or "frame-based" mode, where each batch or frame is a collection of consecutive signal samples that have been buffered into a single unit. By propagating these multisample frames instead of the individual signal samples, the DSP system can best take advantage of the speed of DSP algorithm execution, while simultaneously reducing the demands placed on the data acquisition (DAQ) hardware.
- The DSP Blockset delivers this same high level of performance for both simulation and code generation by incorporating frame-processing capability into all of its blocks. A completely frame-based model can run several times faster than the same model processing sample-by-sample; faster still if data sources are frame based.

DSP Blockset Features

Matrix Support:

The DSP Blockset takes full advantage of the matrix format of Simulink. Some typical uses of matrices in DSP simulations are

- General two-dimensional array: a matrix can be used in its traditional mathematical capacity, as a simple structured array of numbers. Most blocks for general matrix operations are found in the Matrices and Linear Algebra library.
- Factored submatrices : a number of the matrix factorization blocks in the Matrix Factorizations library store the submatrix factors (i.e., lower and upper submatrices) in a single compound matrix. See the LDL Factorization and LU Factorization blocks for examples.
- Multichannel frame-based signal: the standard format for multichannel frame-based data is a matrix containing each channel's data in a separate column. A matrix with three columns, for example, contains three channels of data, one frame per channel. The number of rows in such a matrix is the number of samples in each frame.

DSP Blockset Features

Adaptive and Multirate Filtering:

- The Adaptive Filters and Multirate Filters libraries provide key tools for the construction of advanced DSP systems. Adaptive filter blocks are parameterized to support the rapid tailoring of DSP algorithms to application-specific environments, and effortless "what if" experimentation. The multirate filtering algorithms employ polyphase implementations for efficient simulation and real-time code execution.

DSP Blockset Features

Statistical Operations:

- Use the blocks in the Statistics library for basic statistical analysis. These blocks calculate measures of central tendency and spread (e.g., mean, standard deviation, and so on), as well as the frequency distribution of input values (histograms).

DSP Blockset Features

Linear Algebra:

- The Matrices and Linear Algebra library provides a wide variety of matrix factorization methods, and equation solvers based on these methods. The popular Cholesky, LU, LDL, and QR factorizations are all available.
-
- Uitwerken – wat is autoregressie ..

DSP Blockset Features

Parametric Estimation:

- The Parametric Estimation library provides a number of methods for modeling a signal as the output of an AR system. The methods include the Burg AR Estimator, Covariance AR Estimator, Modified Covariance AR Estimator, and Yule-Walker AR Estimator, which allow you to compute the AR system parameters based on forward error minimization, backward error minimization, or both.

DSP Blockset Features

Real-Time Code Generation (optional) :

- The separate Real-Time Workshop can be utilized to generate ANSI C code for models containing blocks from the DSP Blockset.

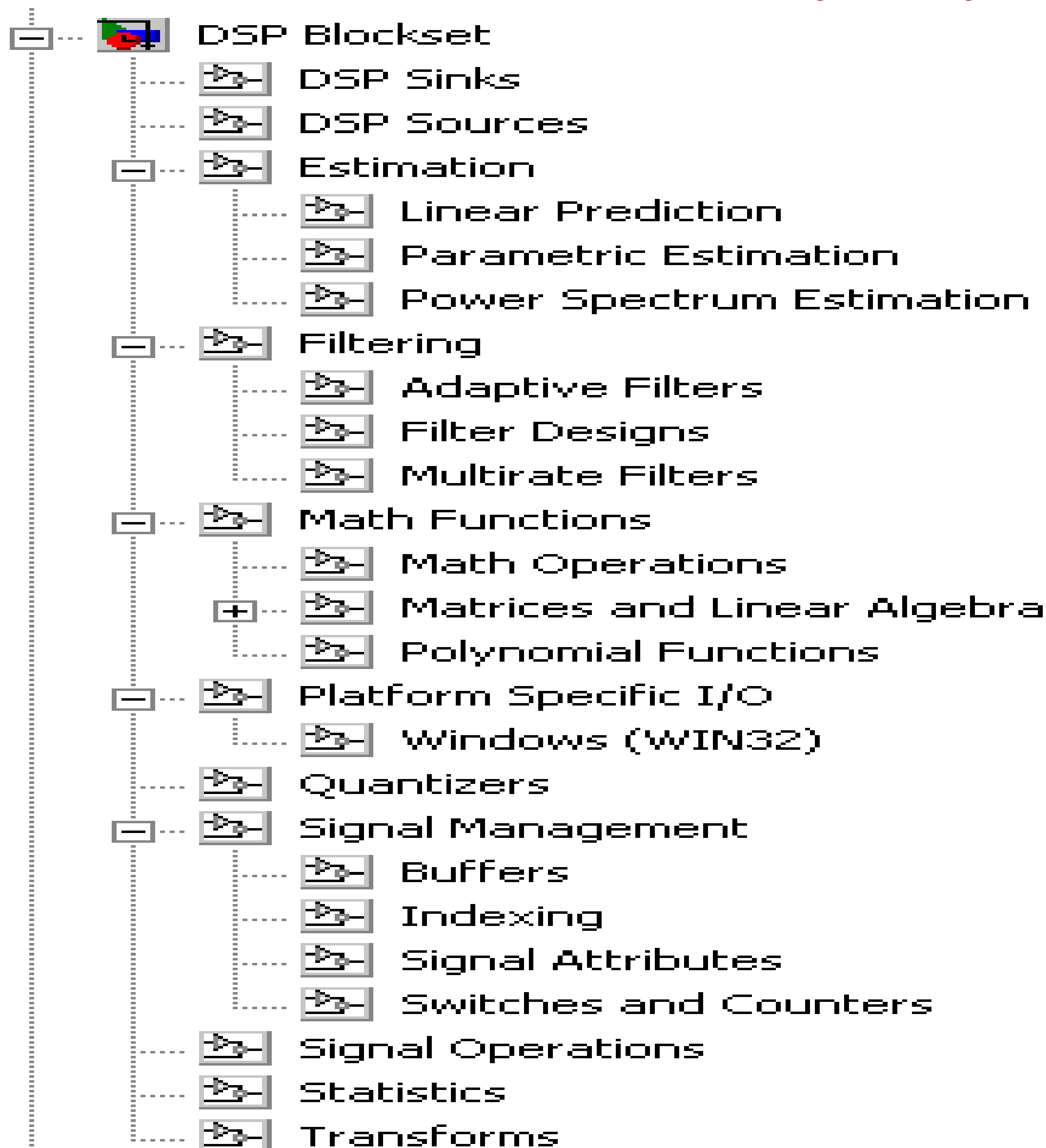
DSP blockset set-up

The DSP Blockset contains a collection of blocks organized in a set of nested libraries.

One way to explore the blockset is to expand the DSP Blockset entry in the Simulink Library Browser.

The other way to commence with the DSP blockset is to enter dsplib at the cmd prompt.

DSP library layout



DSP sources library

Library: dspsrcs4

File Edit View Format Help

DSP Sources

1
DSP Constant

1:10
Signal From Workspace

1:10
Triggered Signal From Workspace

Discrete Impulse

8
N-Sample Enable

4-Phase Clock
Multiphase Clock

Random Source

DSP
Sine Wave

Lin
Chirp

eye(5)
Identity Matrix

Constant Diagonal Matrix

Constant Ramp

Orange blocks support fixed-point data types.

DSP Sources

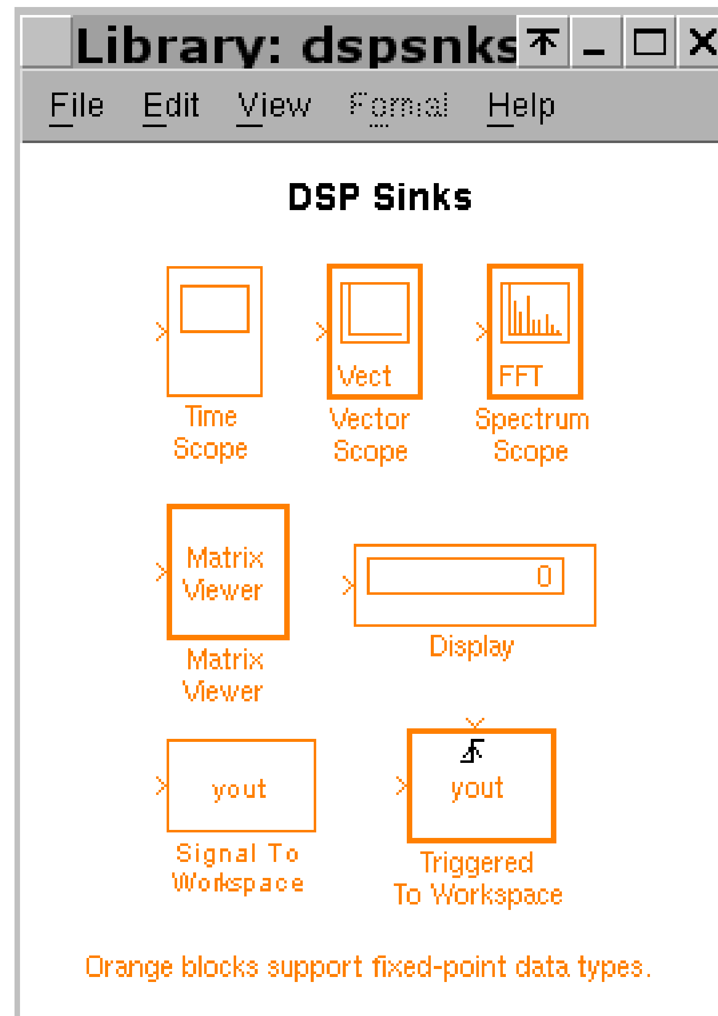
DSP Sources are blocks that generate discrete-time signals such as sine waves and uniform random signals. These signals are tailored towards DSP applications and include:

- *Chirp*: generate a swept-frequency cosine (chirp) signal
- *Constant Diagonal Matrix*: generate a square, diagonal matrix
- *Constant Ramp*: generate a ramp signal with length based on input dimensions
- *Discrete Impulse*: generate a discrete impulse

DSP Sources II

- *DSP Constant*: generate a discrete-time or continuous-time constant signal
- *Identity Matrix*: generate a matrix with ones on the main diagonal and zeros elsewhere
- *Multiphase Clock*: generate multiple binary clock signals
- *N-Sample Enable*: output ones or zeros for a specified number of sample times
- *Random Source*: generate randomly distributed values (Gaussian or uniform)
- *Signal From Workspace*: import a signal from the MATLAB workspace
- *Sine Wave*: generate a continuous or discrete sine wave
- *Triggered Signal From Workspace*: import signal samples from the MATLAB workspace when triggered

DSP Sinks



DSP sinks

DSP Sinks are blocks for exporting signals to the MATLAB workspace:

- *Display* (Simulink block): show the value of the input
- *Matrix Viewer*: display a matrix as a color image
- *Signal To Workspace*: write simulation data to an array in the MATLAB workspace
- *Spectrum Scope*: compute and display the short-time FFT of each input signal
- *Time Scope* (Simulink Block): display signals generated during a simulation
- *Triggered To Workspace*: write the input sample to an array in the MATLAB workspace when triggered
- *Vector Scope*: display a vector or matrix of time-domain, frequency-domain, or user-defined data

Standard filters

Library: dsparch4

File Edit View Format Help

Filter Designs

FDATool
Digital Filter Design

butter
Analog Filter Design

FDATool
Filter Realization Wizard

Filter Implementations

IIR DF2T
Digital Filter

Overlap Save
Overlap-Save FFT Filter

Overlap Add
Overlap-Add FFT Filter

Orange blocks support fixed-point data types.

Standard analog and digital filters

- These DSP blocks are intened for designing, analyzing, and implementing various standard filters.
- *Analog Filter Design*: Design and implement an analog filter
- *Digital Filter*: Filter inputs with a specified time-varying or static digital FIR or IIR filter
- *Digital Filter Design*: Design, analyze, and implement a variety of digital FIR and IIR filters.
- *Filter Realization Wizard*: Automatically construct filter realizations using Sum, Gain, and Unit Delay blocks
- *Overlap-Add FFT Filter*: Implement the overlap-add method of frequency-domain filtering
- *Overlap-Save FFT Filter*: Implement the overlap-save method of frequency-domain filtering

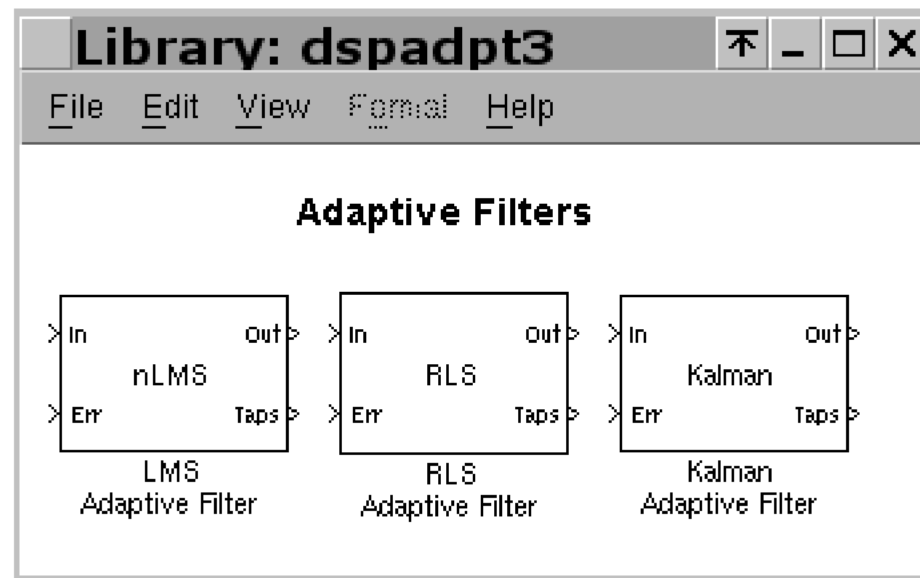
Exercise 2

- The analog filter designed in exercise 1 must be validated in a simulation.
- Take the analogue filter requirements and implement them in a Simulink model.

(hint: use the DSP filter realization tools)

- Implement your filter into a Simulink model and insert a heart-beat signal (25 Hz) with added white noise and a linear disturbance at 70 Hz. Validate your filter design by examining the filter output.

Adaptive Filters

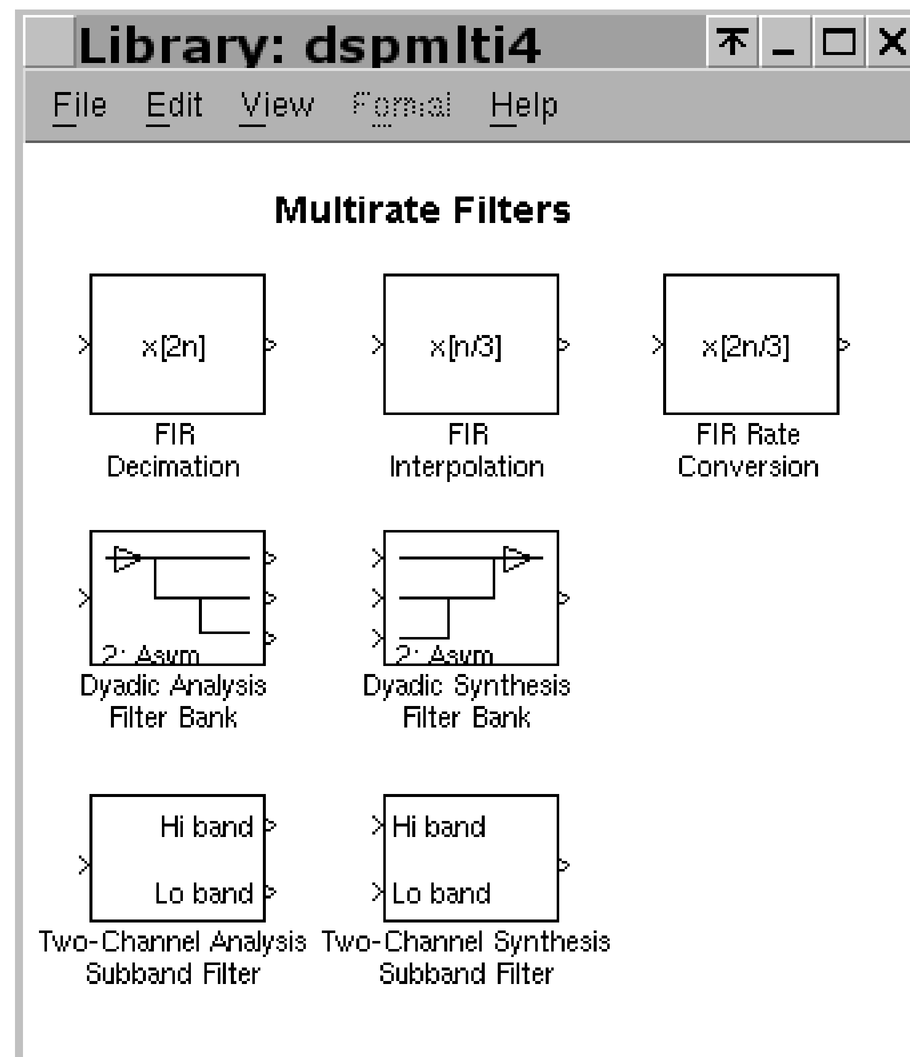


Adaptive filters

adaptive filters blocks are used for computing filter estimates of an input using various algorithms:

- *Kalman Adaptive Filter*: compute filter estimates for an input using the Kalman adaptive filter algorithm
- *LMS Adaptive Filter*: compute filter estimates for an input using the LMS adaptive filter algorithm
- *RLS Adaptive Filter*: compute filter estimates for an input using the RLS adaptive filter algorithm

Multirate filters



Multirate filters

Multirate Filters blocks are used when implementing various multirate filters:

- *Dyadic Analysis Filter Bank*: decompose a signal into components of equal or logarithmically decreasing frequency subbands and sample rates
- *Dyadic Synthesis Filter Bank*: reconstruct a signal from its multirate bandlimited components.
- *FIR Decimation*: filter and downsample an input signal
- *FIR Interpolation*: upsample and filter an input signal

Multirate filtersII

- *FIR Rate Conversion*: upsample, filter, and downsample an input signal
- *Two-Channel Analysis Subband Filter*: decompose a signal into a high-frequency subband and a low-frequency subband
- *Two-Channel Synthesis Subband Filter*: reconstruct a signal from a high-frequency subband and a low-frequency subband

Multirate filter example

- Please visit the 'multirate filtering' section of the DSP Blockset demos to see a few examples of this technique

End

Questions ?

End of session 7