FH JOANNEUM – University of Applied Sciences
Elektronik and Computer Engineering
Florian Mayer

Applied Signal Processing
Discrete-Time Signals and Systems
Computer Laboratory 1 & 2

Please submit a single zip-File (and just use zip) including your documentation and all simulation files, e.g., MATLAB files (*.m). You have to zip (and use only zip) all these files to one single file.

Please make sure that your approaches, procedures and results are clearly presented.
!! For all exercises in which audio examples are played, start with a low volume!!

Example 1: Special Signals

Investigate in the MATLAB functions `ones, zeros, find` using MATLAB help

Special Signals play a crucial role in DSP, they are used to determine the impulse response and support the representation of signals. In this very first task implement the delta-function and the step function as standalone functions.

$$\delta[n] = \begin{cases} 1, & \text{if } n = 0, \\ 0, & \text{if } n \neq 0. \end{cases}$$

$$u[n] = \begin{cases} 1, & \text{if } n \geq 0, \\ 0, & \text{if } n < 0. \end{cases}$$

After your implementation test your functions and create and `stem` the following signals:

(a) $\delta[n+1]$, $\delta[n-3]$ and $\delta[n+6]$

(b) $u[n+5]$, $u[n]$ and $u[n-3]$

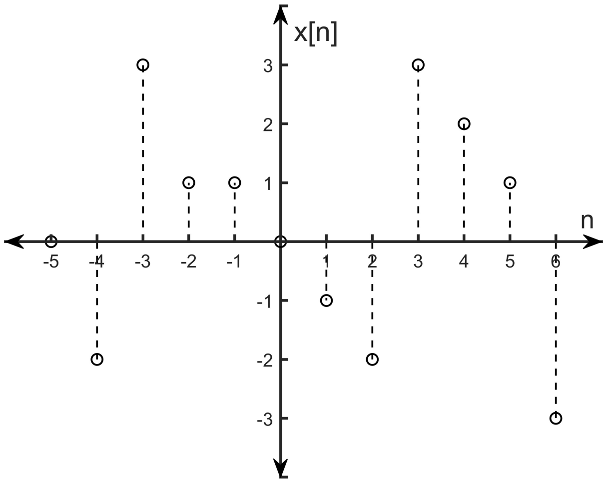
(c) $3\delta[n+1] + n^2(u[n+5] - u[n+4]) + 10(0.5)^n$

for $-20 \leq n \leq 20$

Example 2: Signal modification

Investigate in the MATLAB functions `fliplr, circshift` using MATLAB help

Modify the following signal:



After your implementation test your functions and create and `stem` the following signals:

(a) $x[3n+5]$ **(b)** $x[-5n+4]$ **(c)** $x[n^2]$ **(d)** $y[n] = x[n] - u[n-1]x[n-2]$

Example 3: Representation of DT Signals

Investigate in the MATLAB functions `imag, real, fliplr, circshift` using MATLAB help

Use the signal $x[n] = (-0.2 + j0.3)^n \sin(\frac{\pi}{4}n)(u[n+3] - u[n-5])$ for the following tasks.

(a) Write a function `[vxe,vxo]=analyzeSignal(vn,vx)`, which calculates the even $x_e[n]$ and the odd $x_o[n]$ part of the signal $x[n]$.

(b) Plot the real part and the imaginary part of $x_e[n]$. Does the result agree with your expectations?

(c) Plot the real part and the imaginary part of $x_o[n]$. Does the result agree with your expectations?

(d) Apply the function `analyzeSignal(vn,vx)` to the signals $x_r[n] = \Re(x[n])$ and $jx_i[n] = j\Im(x[n])$.

(e) Plot the signals $x_{r,e}[n]$ and $x_{r,o}[n]$. Do the results agree with (b) and (c)? Why or why not?

(f) Plot the signals $x_{i,e}[n]$ and $x_{i,o}[n]$. Do the results agree with (b) and (c)? Why or why not?

Example 4: Convolution

Investigate in the MATLAB functions `audioread, sound` using MATLAB help

For a given input signal $x[n] = [1, 5, -1, 6, 3]$ and a system output signal $y[n] = 3x[n] - 4x[n-2] + 6x[n-4]$ compute the convolution of two vectors:

(a) Determine the impulse respons $h[n]$ of the given system.

(b) Write a function `[vconOut] = convASP(vInput1, vInput2)`, which calculates the convolution of the input vectors. Compare your results with the built-in MATLAB function `conv`.

(c) Generate the the following impulse response $h_{ech}[n] = \sum_{k=0}^{\infty} \alpha^k \delta(n - kN)$. For $\alpha = 0.7$ and $N = 4$ for for $-20 \leq n \leq 20$.

(d) Load the audiofile `aclarinet.mp3` (Moodle) into your workspace using `audioread` and compute the convolution with the generated impulse response h_{ech}. Set N to a number of samples for one second. Play the result using the MATLAB function `sound`. What do you notice? After that, set the values α and N to your own desire.

Example 5: Sounds

Investigate in the MATLAB functions `linspace, sum, sin` using MATLAB help

The creation of signals or sounds is sometimes a great approach to test computed systems or just to have fun. In order to generate *oscillating* sounds we need to consider the following.

- The sampling frequency f_s of a signal let us consider $f_s = 16384Hz$
- A proper duration vector t in order to create time instances for each sample $t = [0 : \Delta t : T]$ where the length of t is the duration of the tone multiplied by the number of samples for one second.
- In order to create an oscillating signal we need an oscillator ω as well.

(a) Create the following sounds at $440Hz$ for a sampling frequency $f_s = 16384Hz$ with a duration of $T = 0.5s$:

$$\text{Sine wave: } y(t) = sin(wt)$$

$$\text{Square wave: } y(t) = \begin{cases} 0.6, & \text{if } sin(wt) > 0, \\ 0, & \text{if } sin(wt) < 0. \end{cases}$$

$$\text{3 sine waves: } y(t) = 0.6sin(wt) + 0.3sin(2wt) + 0.1sin(4wt)$$

`Plot` each sound and listen to it, if you want to, using the function `sound`. How could a triangular wave be implemented? **Hint**: Use the Fourier_JavaApplet to see which cosines and sines, as well as their harmonics, are used to create the signal.

(b) Download the `FlowSynth.m` file from Moodle, investigate some time to see what happens there. Place your generated sounds at the proper position in the code in order to be used in the synthesizer GUI. Is it possible to add the method from Example 4 as well? What needs to be modified for a practical use?

(c) Play along if you want to and Have fun!

Example 6: Pre-Processing filter

Investigate in the MATLAB functions `filter` using the MATLAB help

Smoothing is how we discover important patterns in our data while leaving out things that are unimportant (i.e. noise). We use filtering to perform this smoothing. The goal of smoothing is to produce slow changes in value so that it's easier to see trends in our data.

$$\text{Moving Average Filter:} \quad y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[n-k] \tag{1}$$

$$\text{Median Filter:} \quad \widetilde{x} = (x_1, x_2, ..., x_n) \quad y[n] = \begin{cases} \widetilde{x}[\frac{n+L+1}{2}] & \text{if } n \text{ is odd} \\ \frac{1}{2}(\widetilde{x}_{\frac{n+L}{2}} + \widetilde{x}_{\frac{n+L+1}{2}}) & \text{if } n \text{ is even} \end{cases} \tag{2}$$

$$\text{Weighted Moving Average Filter:} \quad y[n] = \sum_{n=-k}^{k} wx[n-k] \quad \text{where} \quad w = \frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8} \tag{3}$$

Listing 1: MATLAB built-in test data

```
load bostemp
days = (1:31*24)/24;
plot(days, tempC)
axis tight
ylabel('Temp (\circC)')
xlabel('Time elapsed from Jan 1, 2011 (days)')
title('Logan Airport Dry Bulb Temperature (source: NOAA)')
```

(a) The moving average filter is a simple Low Pass FIR (Finite Impulse Response) filter. It takes L samples of input at a time and takes the average of those L-samples and produces a single output point. It is a very simple LPF (Low Pass Filter) structure that comes handy for scientists and engineers to filter unwanted noisy component from the intended data.
Create a function `myMovingAverage(signal, sampleLength)` in order to smooth the measurement results. Adapt and compare your results with the following:

Listing 2: Moving average filter

```
hoursPerDay = 24;
coeff24hMA = ones(1, hoursPerDay)/hoursPerDay;
avg24hTempC = filter(coeff24hMA, 1, tempC);
plot(days,[tempC avg24hTempC])
```

(b) Median filters are useful in reducing random noise, especially when the noise amplitude probability density has large tails, and periodic patterns. The median filtering process is accomplished by sliding a window over the image. The filtered image is obtained by placing the median of the values in the input window, at the location of the center of that window. Create a function `myMedianFilt(signal, sampleLength)` in order to smooth the measurement results.

(c) A major advantage of weighted moving averages is that they yield a smoother estimate of the trend-cycle. Instead of observations entering and leaving the calculation at full weight, their weights slowly increase and then slowly decrease, resulting in a smoother curve.
Create a function `myWeightedMA(signal, sampleLength, weights)` in order to smooth the measurement results. It is important that the weights all sum to one and that they are symmetric so that $w_{-k} = w_k$, choose your own weights. Adapt and compare your results with the following:

Listing 3: Weighted Moving average filter

```
h = [1/2 1/2];
binomialCoeff = conv(h,h);
for n = 1:4
        binomialCoeff = conv(binomialCoeff,h);
end
figure
fDelay = (length(binomialCoeff)-1)/2;
binomialMA = filter(binomialCoeff, 1, tempC);
plot(days,tempC,days-fDelay/24,binomialMA)
```

(d) Compare your implemented filter results, which one fits the best?

Example 7: Discrete-Time Fourier Transform - Implementation

(a) Implement the discrete-time Fourier transform `[vX]=computeDTFT(vn,vx,vw)` without using a loop. Consider the general form of the DTFT:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

(b) Compute the DTFT of the input signal of Example 4 using your function `computeDTFT`. Plot your result. Does the result agree with your expectations? Why or why not?

(c) Compare your function with the MATLAB command `fft` for the signal $x[n]$. Do the results agree? What are the restrictions on n and ω_k? How can you modify the signals or the results so that both calculations agree? You can use the command `norm(vxX1-vx2,2)` to calculate the Euclidean norm between the two results (see the MATLAB reference pages for details).