MATLAB

(0:1:length(signal)-1)/f_s defines the time vector of the signal sampled at f_s

downsample(signal, f_s/f_d) return the signal, originally sampled at f_s, downsampled at f_ds

stem(n,y) plots the data sequence y at values specified by n

trapz(y) computes the approximate integral of y via the trapezoidal method

interp1(x,y,xq) return interpolated values at the query points xq of the signal defined by the values y at the sample points x

zplane(b,a) displays the z-plot of the transfer function with **b** and **a** coefficients at the **numerator** and at the **denominator** respectively.

[H,w]=freqz(b,a,n) returns the n-point frequency response **H** (DFT length is n) of the transfer function with coefficients b and a and the respective frequencies w, ranging from 0 to π (excluded). Additional parameter "whole" return the DFT from 0 to 2π excluded.
abs(**H**) and angle(**H**) can be used to return the magnitude and the phase of the frequency response **H.** In order to have a plot we can simply right the code without **[H,w]=**

rectwin(M), bartlett(M), hann(M), hamming(M) and blackman(M) return M-points windows

X=**fft**(**x**,n) is the fast Fourier Transform of n points; if length(x)<n, then x is zero-padded to n.

Usually n is a power of 2 in order to exploit the fft algorithm.

x=**ifft**(**X**,n) is the inverse fast Fourier Transform of n points; y is zero-padded to length n

y=filter(b,a,x) performs digital filtering of the input signal x with the filter defined by the transfer function coefficients b and a

y=filtfilt(**b**,**a**,**x**) performs zero-phase digital filtering of the input x with the filter whose transfer function coefficients are b and a (actual filter order is double the order of the filter specified by b and a)

b=fir1(n,fcut,ftype) it's a function for design a fir filter (using the windowing method) of the ftype (can be "stop", "low","high" or "pass") with a number n of coefficients (length of the IR)
if i want a notch filter we can put as a parameters:
ftype="stop"
fcut=[49/fn,51/fn] in this function we need normalized frequencies wrt the **nyquist frequency**.
We can also add a parameter "window" if we want to design the filter with a specific window different from the rectangular one.

[b,a]=butter(N,fcut,ftype) returns the transfer function coefficients of an order N lowpass digital Butterworth filter with cut-off frequency fcut (which must be normalized over the Nyquist frequency) . In ftype it's possible to specifiy another type of filter (lowpass is the default one).

[b,a]=cheby1(N,Rp,fcut,ftype) design a type1 cheby filter (fcut is a scalar or a vector of two element, in which we put the normalized (fn) cut-off frequency of the filter. **Rp** represents the peak-to-peak passband ripple in decibels. If "ftype" parameter is not included at the end as a parameter we obtain an LP filter of order **N**.

Eg.

Design a 2N=2*3=6 6th-order Chebyshev Type I bandstop filter with normalized edge frequencies of 0.2π and 0.6π rad/sample and 5 dB of passband ripple. Plot its magnitude and phase responses. Use it to filter random data.

```
[b,a] = cheby1(3,5,[0.2 0.6],'stop');
```

[b,a]=cheby2(N,As,fcut,ftype)

[b,a]=ellip(N,Rp,Rs,fcut,ftype) return the transfer function coefficients of order N lowpass digital filters with cut-off frequency fcut where Rp is the passband ripple and As is the stopband attenuation (both in decibels)

Eg. Design a 6th-order lowpass elliptic filter with 10 dB of passband ripple, 50 dB of stopband attenuation, and a passband edge frequency of 300 Hz, which, for data sampled at 1000 Hz, corresponds to 0.6π rad/sample

```
fcut = 300;
fs = 1000;

[b,a] = ellip(6,10,50,fcut/(fs/2));
```

[b,a]=iirnotch(wnotch ,bw) returns the transfer function coefficients of a second-order IIR notch (band-stop) filter with the notch set at the normalized frequency wnotch and the bandwidth at the -3 dB point set at bw (how much of the surrounding frequencies you want to affect)

Eg. Design and plot an IIR notch filter that removes a 60 Hz tone (f0) from a signal at 300 Hz (fs). For this example, set the Q factor for the filter to 35 and use it to specify the filter bandwidth.

```
wo = 60/(300/2);
bw = wo/35; QUALITY FACTOR
[b,a] = iirnotch(wo,bw);
```

[**pxx**,f]=**periodogram**(xn,*window*,nfft,fs) returns the **periodogram (PSD estimate) pxx** of **the input signal xn** sampled at frequency fs, using the *specified window* and using nfft points in the DFT(returns also the corresponding vector of frequencies f). The default window is the rectangular window. In all these function nfft for default is 256 or the NEXT power of 2 greater than length(xn), in order to apply the FFT.

Eg.

Obtain the periodogram of an input signal consisting of a discrete-time sinusoid with an angular frequency of $\pi/4$ radians/sample with additive $N(0,1)$ white noise. Use a DFT length equal to the signal length.

Create a sine wave with an angular frequency of $\pi/4$ radians/sample with additive $N(0,1)$ white noise. The signal is 320 samples in length. Obtain the periodogram using the default rectangular window and DFT length equal to the signal length. Because the signal is real-valued, the one-sided periodogram is returned by default with a length equal to 320/2+1.

Get

```
n = 0:319;
x = cos(pi/4*n)+randn(size(n));
nfft = length(x);
periodogram(x,[],nfft)
```

[**pxx**,f]=**pwelch** (xn,*window*,**numoverlap**,nfft,fs) displays the PSD estimate pxx of the input signal xn using Welch's method with **numoverlap** samples of overlap from segment to segment (default value for numoverlap is ½ length(xn) the default window is the hamming window). The length of the window specify the number of sample for each window, aka M.

Eg.

```
N=1500;%samples Fs=500;%frequency of sampling
n_overlap=0;
M=Fs*10; %(we have 5000 samples, M=500, so N/M=K=3 non overlapping windows)
[Pxx,f] = pwelch(x,rectwin(M),n_overlap,[],Fs); %k=3 windows(30 seconds is the
complete time window)
% - Bartlett (10 seconds window, so Fs*10seconds=M is the number of sample in
the window) %in this case
```

[pxy,f]=**cpsd**(xn,yn,window,**numoverlap**,nfft,fs) returns the cross-PSD estimate of xn and yn using **Welch's averaged periodogram method**

[kxy,f]=**mscohere**(xn,**yn**,window,numoverlap,nfft,fs) returns the magnitude-squared coherence estimate kxy of xn and yn

Eg.

Estimate the magnitude-squared coherence of x and y. Use a 512-sample Hamming window. Specify 500 samples of overlap between adjoining segments and 2048 DFT points.

```
[cxy,fc] = mscohere(x,y,hamming(512),500,2048);
```

[pxx,f]=pyulear(xn,**order**,nfft,fs) returns the PSD estimate pxx (using nfft points, which means that pxx has length (nfft/2+1) if nfft is even, and (nfft+1)/2 if nfft is odd-------default is 256) using the nth-**order** Yule-Walker method and

returns the corresponding vector of frequencies f

[pxx,f]=pburg (xn,**order**,nfft,fs) use the Burg's method

[pxx,f]=pcov(xn,order,nfft,fs) returns … the nth-order covariance method and…

All these functions only display the corresponding estimate if no output arguments are provided