Digital Signal Processing

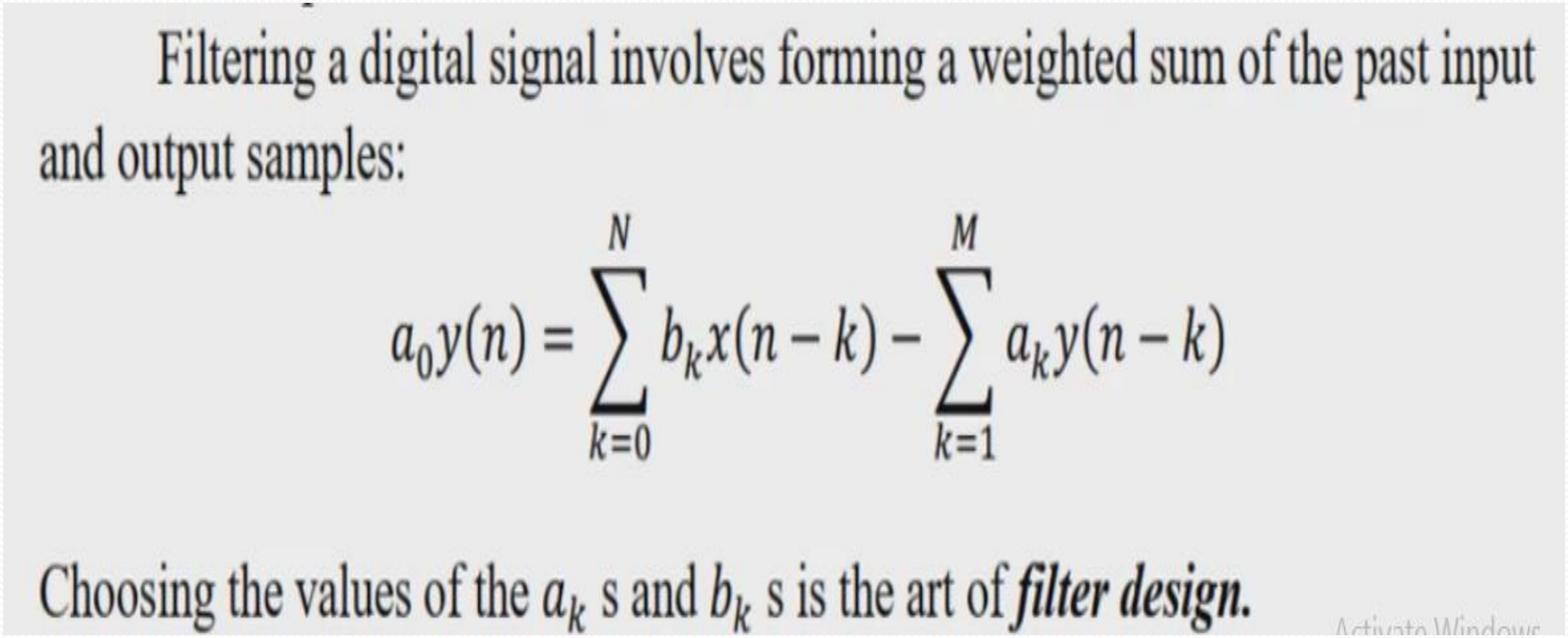
Lab7: Digital Filters

Instructor: Eng\ Samar Shaaban
E-mail: ssa10@fayoum.edu.eg
Github Repo: https://github.com/SamarShabanCS/DSP
Slack workspace: https://fayoum-university-fci.slack.com

Simple filtering:

In digital signal processing applications, it is often necessary to change the relative amplitudes of frequency components or remove undesired frequencies of a signal. This process is called filtering. Digital filters are used in a variety of applications. We saw that digital filters may be used in systems that perform interpolation and decimation on discrete-time signals. Digital filters are also used in audio systems that allow the listener to adjust the bass (low-frequency energy) and the treble (high frequency energy) of audio signals.

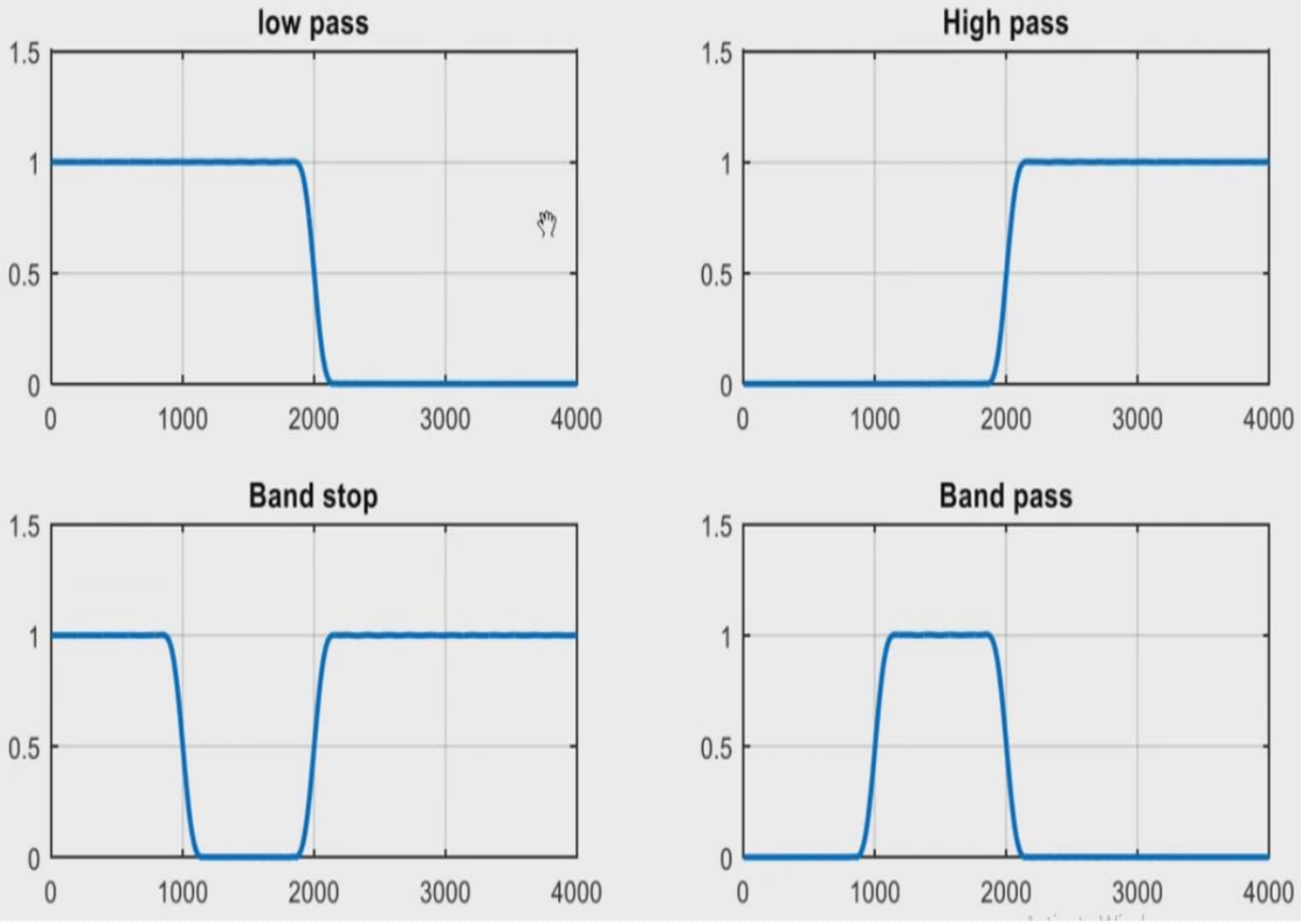
Digital filter design requires the use of both frequency domain and time domain techniques. This is because filter design specifications are often given in the frequency domain, but filters are usually implemented in the time domain with a difference equation.

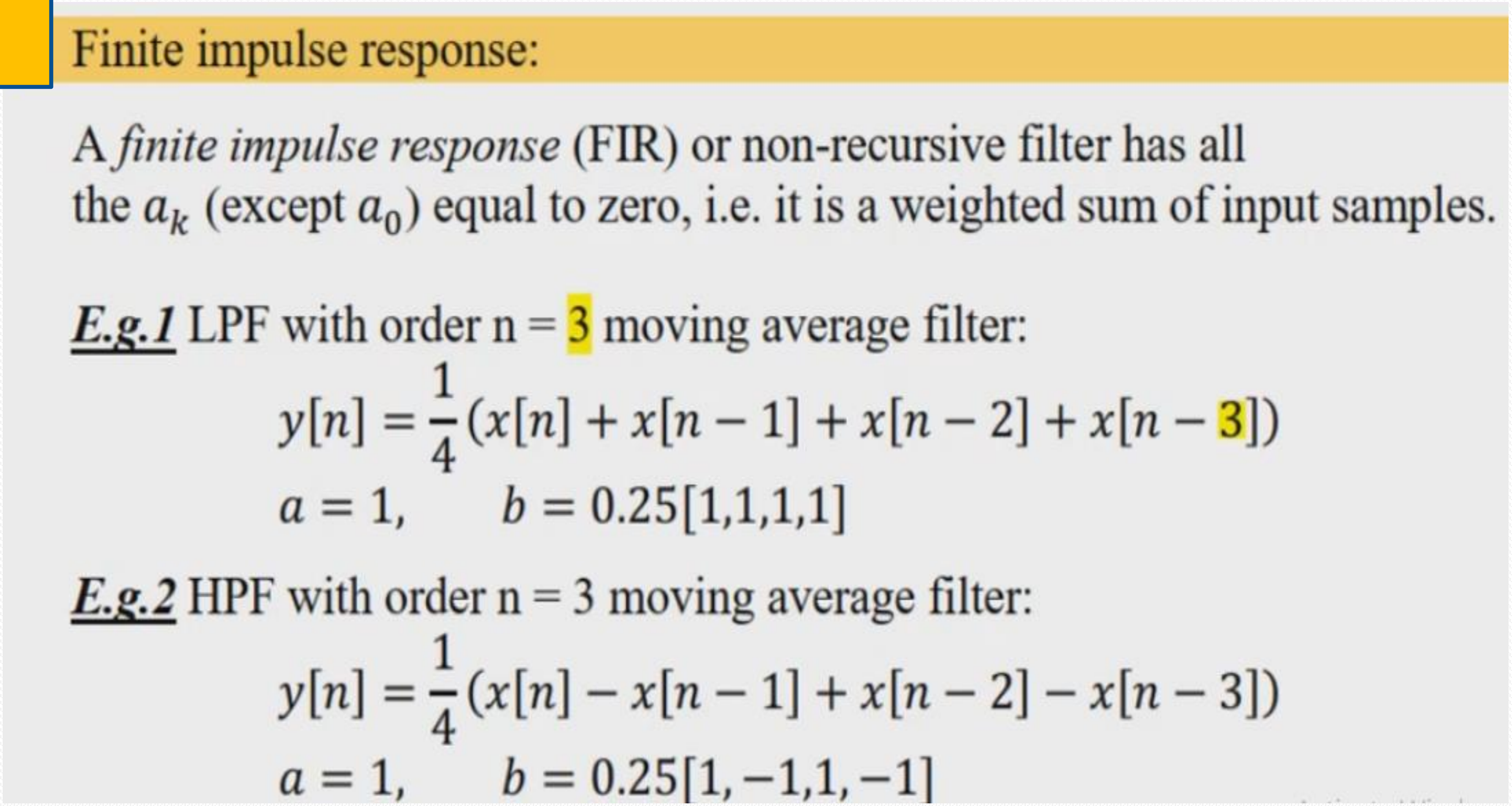


Filtering a digital signal involves forming a weighted sum of the past input and output samples:

$$a_0 y(n) = \sum_{k=0}^{N} b_k x(n-k) - \sum_{k=1}^{M} a_k y(n-k)$$

Choosing the values of the a_k s and b_k s is the art of ***filter design.***


low pass
High pass
Band stop
Band pass
1.5
1
0.5
0
0
1000
2000
3000
4000



Finite impulse response:

A *finite impulse response* (FIR) or non-recursive filter has all the a_k (except a_0) equal to zero, i.e. it is a weighted sum of input samples.

E.g.1 LPF with order n = 3 moving average filter:

$$y[n] = \frac{1}{4}(x[n] + x[n-1] + x[n-2] + x[n-3])$$

$$a = 1, \qquad b = 0.25[1,1,1,1]$$

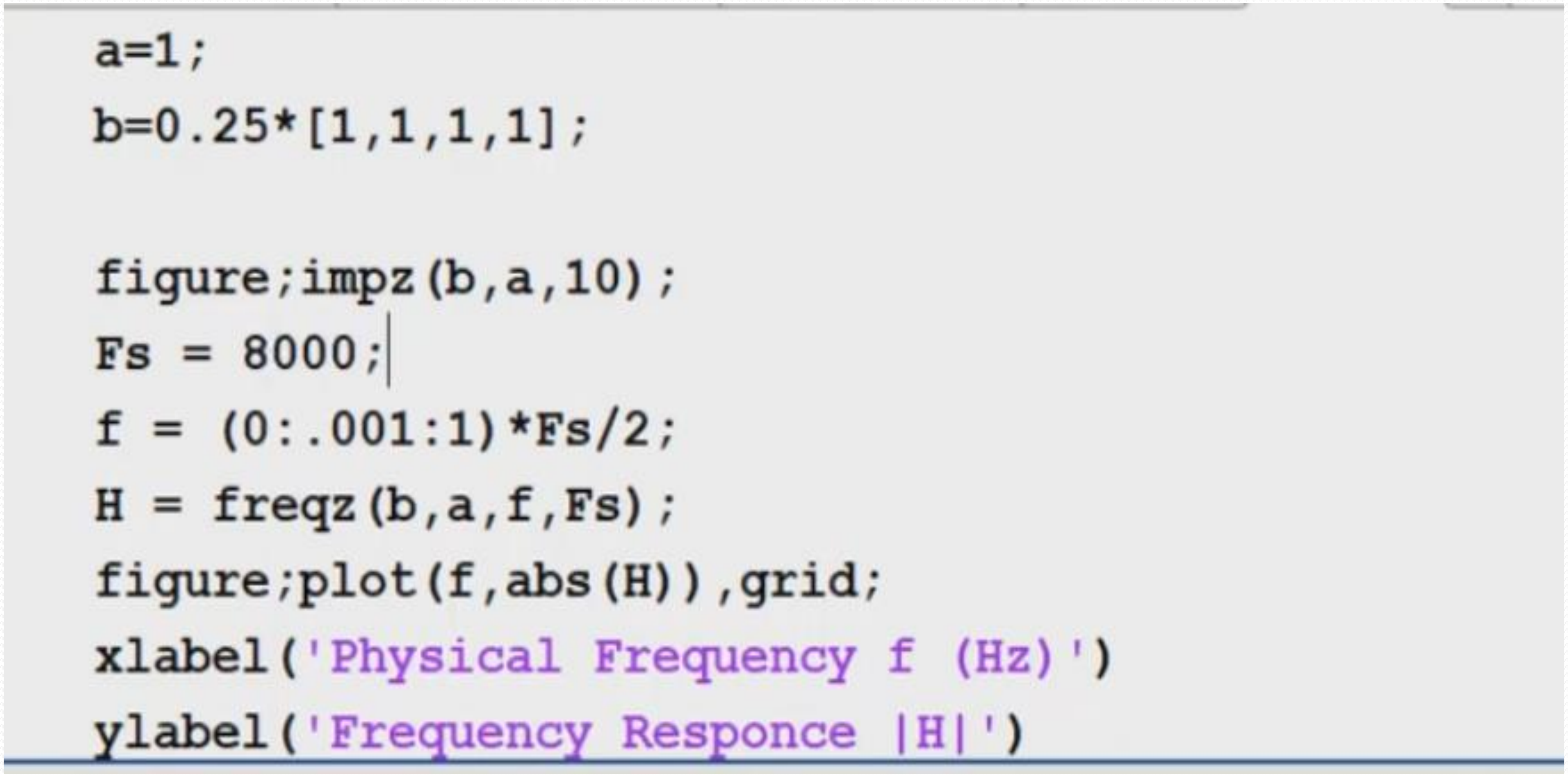
E.g.2 HPF with order n = 3 moving average filter:

$$y[n] = \frac{1}{4}(x[n] - x[n-1] + x[n-2] - x[n-3])$$

$$a = 1, \qquad b = 0.25[1,-1,1,-1]$$

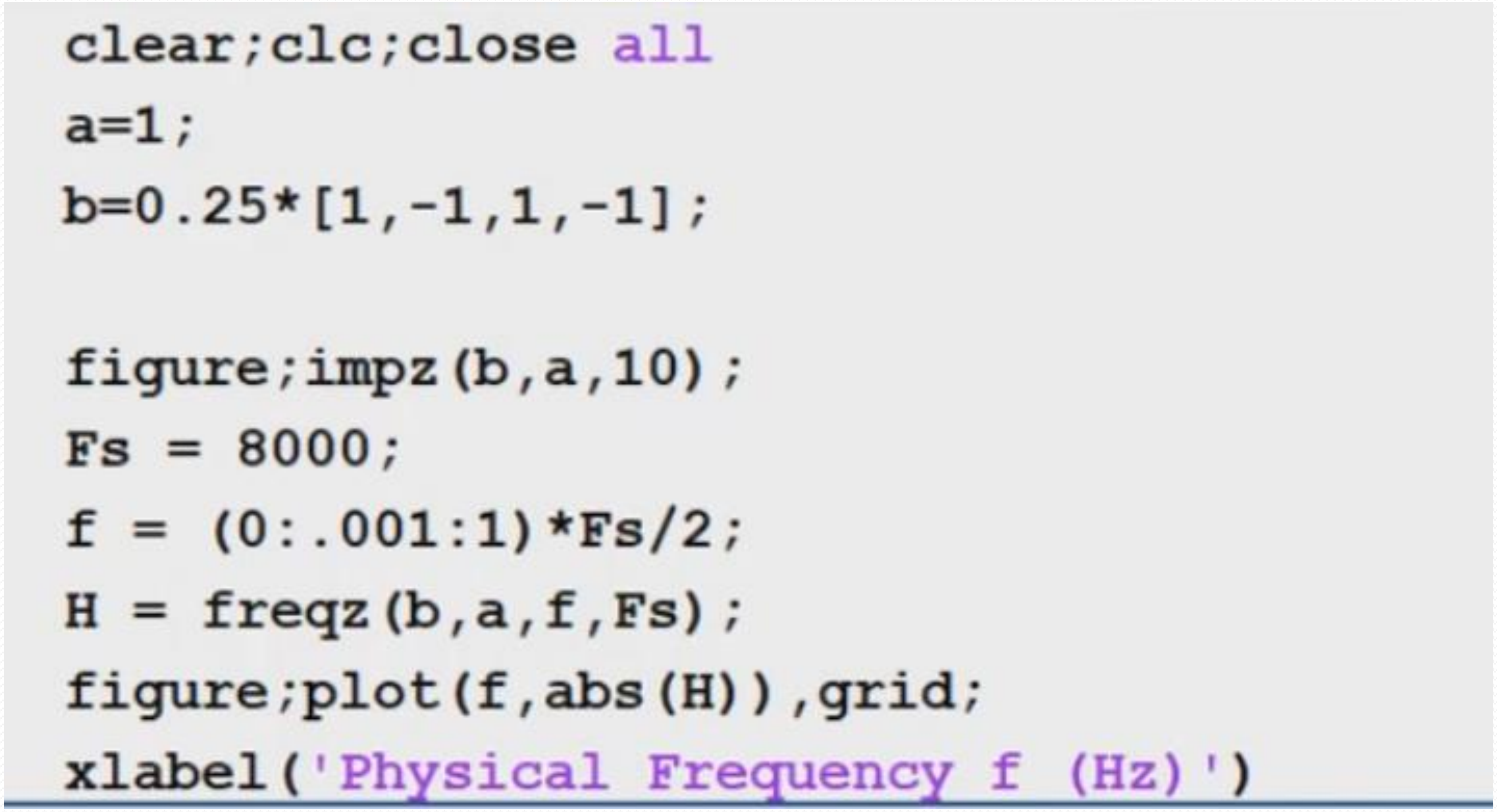
```
a=1;
b=0.25*[1,1,1,1];


figure;impz(b,a,10);


figure;freqz(b,a);
```

Low pass



```
a=1;
b=0.25*[1,1,1,1];

figure;impz(b,a,10);
Fs = 8000;
f = (0:.001:1)*Fs/2;
H = freqz(b,a,f,Fs);
figure;plot(f,abs(H)),grid;
xlabel('Physical Frequency f (Hz)')
ylabel('Frequency Responce |H|')
```

High pass



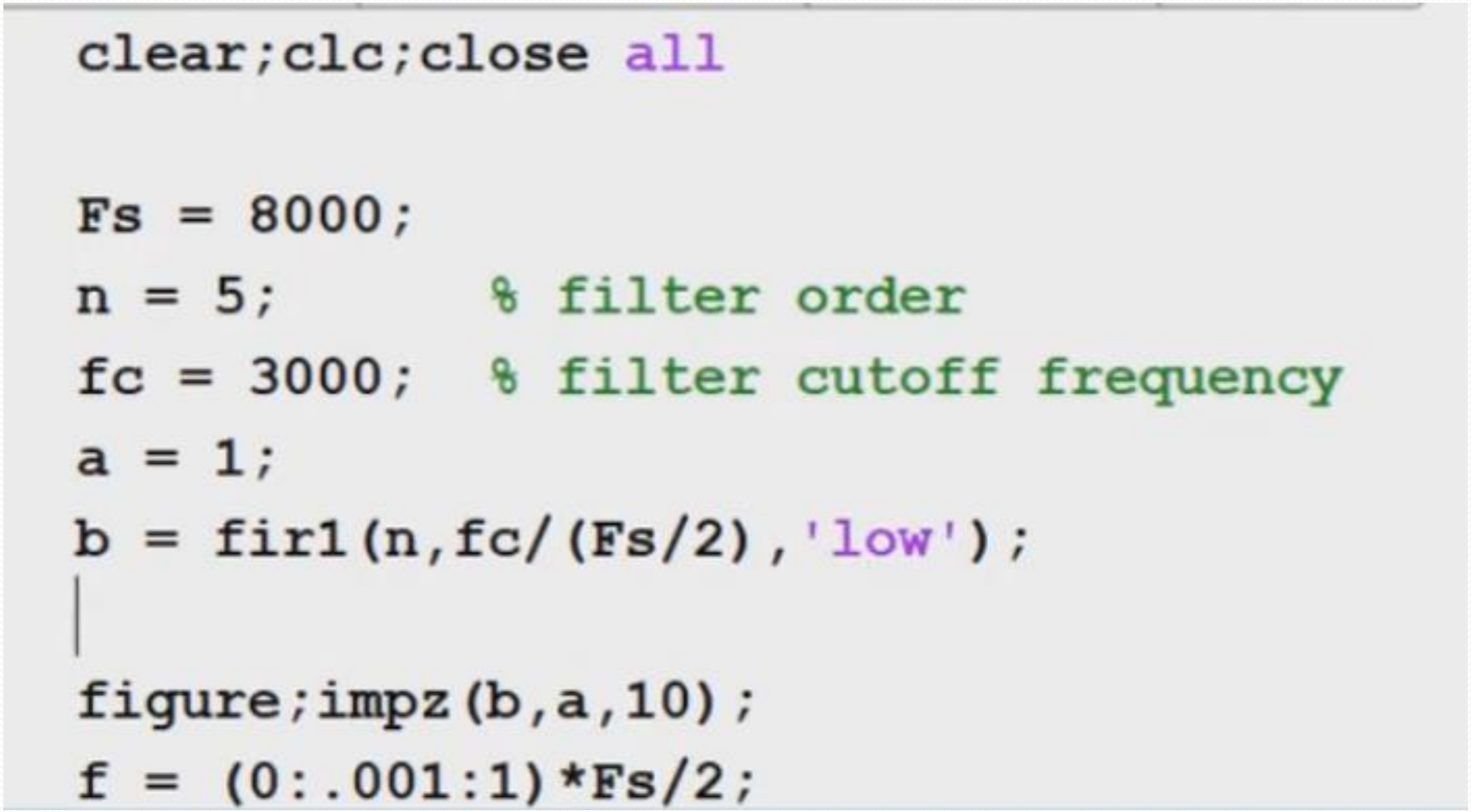
```
clear;clc;close all
a=1;
b=0.25*[1,-1,1,-1];

figure;impz(b,a,10);
Fs = 8000;
f = (0:.001:1)*Fs/2;
H = freqz(b,a,f,Fs);
figure;plot(f,abs(H)),grid;
xlabel('Physical Frequency f (Hz)')
```

The code below calculate best b coefficient for FIR low pass filter:

```
n = 5;                  % filter order
fc = 3000;              % filter cutoff frequency
a = 1;
% Choose filter type 'low','high','bandpass','stop'
b = fir1(n,fc/(Fs/2),'low');
%b = fir1(n,fc/(Fs/2),'high');
%b = fir1(n,[500 1500]/(Fs/2),'bandpass');
%b = fir1(n,[500 1500]/(Fs/2),'stop');
```

After running MATLAB script we get the following equation:

$$y[n] = -0.0039 * x[n] - 0.0323 * x[n-1] + 0.5362 * x[n-2] + 0.5362 * x[n-3] - 0.0323 * x[n-4] - 0.0039 * x[n-5]$$



```
clear;clc;close all


Fs = 8000;
n = 5;          % filter order
fc = 3000;      % filter cutoff frequency
a = 1;
b = fir1(n,fc/(Fs/2),'low');

figure;impz(b,a,10);
f = (0:.001:1)*Fs/2;
```

Infinite impulse response:

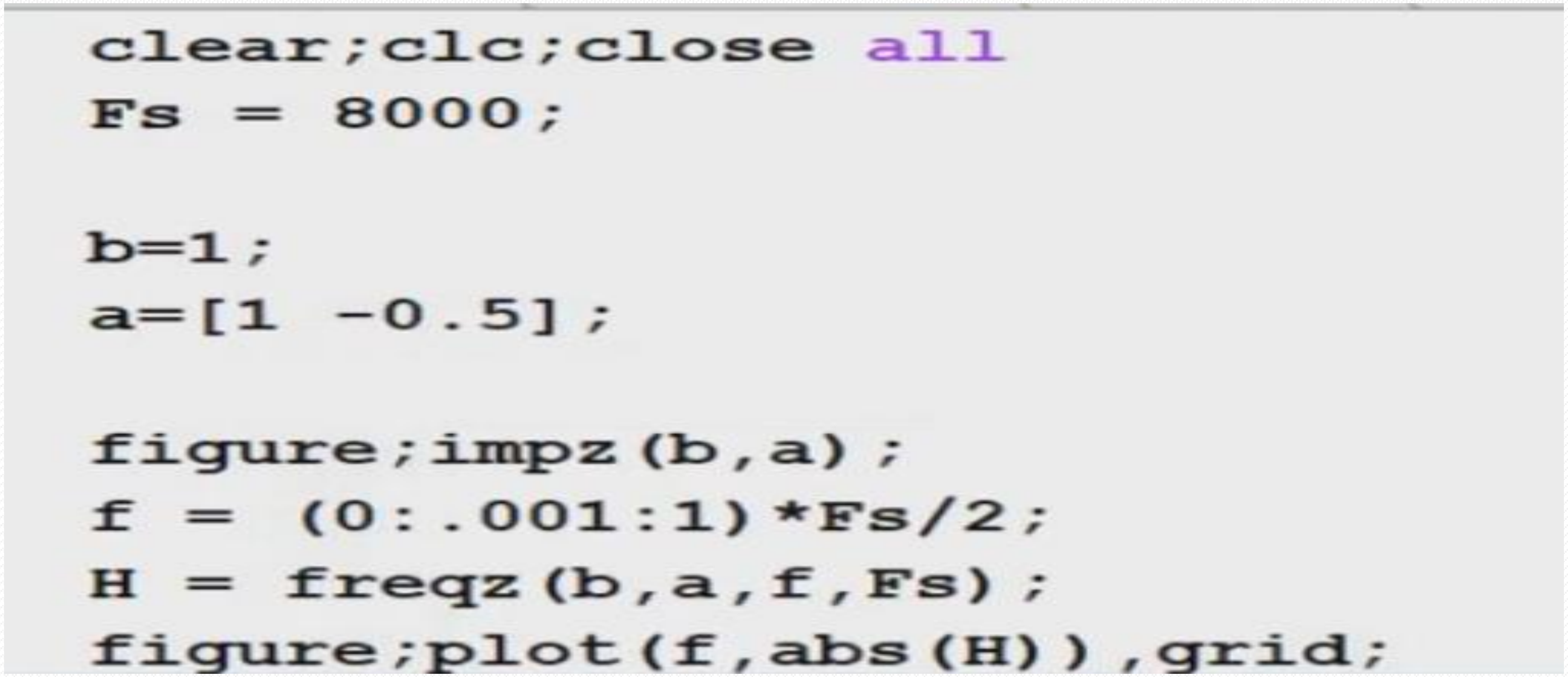
An *infinite impulse response* (IIR) filter has a unit-sample response that doesn't go to zero after a finite number of samples. IIR filters are usually implemented *recursively*, i.e. with non-zero a_k coefficients

E.g.1 LPF with 1st order

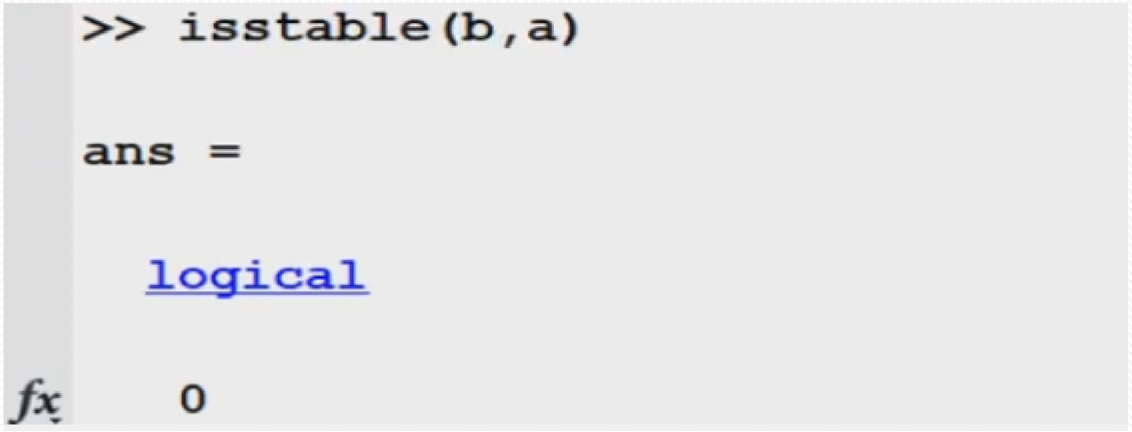
$$y[n] = x[n] + 0.5 * y[n-1]$$

E.g.2 HPF with 1st order

$$y[n] = x[n] - 0.9 * y[n-1]$$



```
clear;clc;close all
Fs = 8000;

b=1;
a=[1 -0.5];

figure;impz(b,a);
f = (0:.001:1)*Fs/2;
H = freqz(b,a,f,Fs);
figure;plot(f,abs(H)),grid;
```



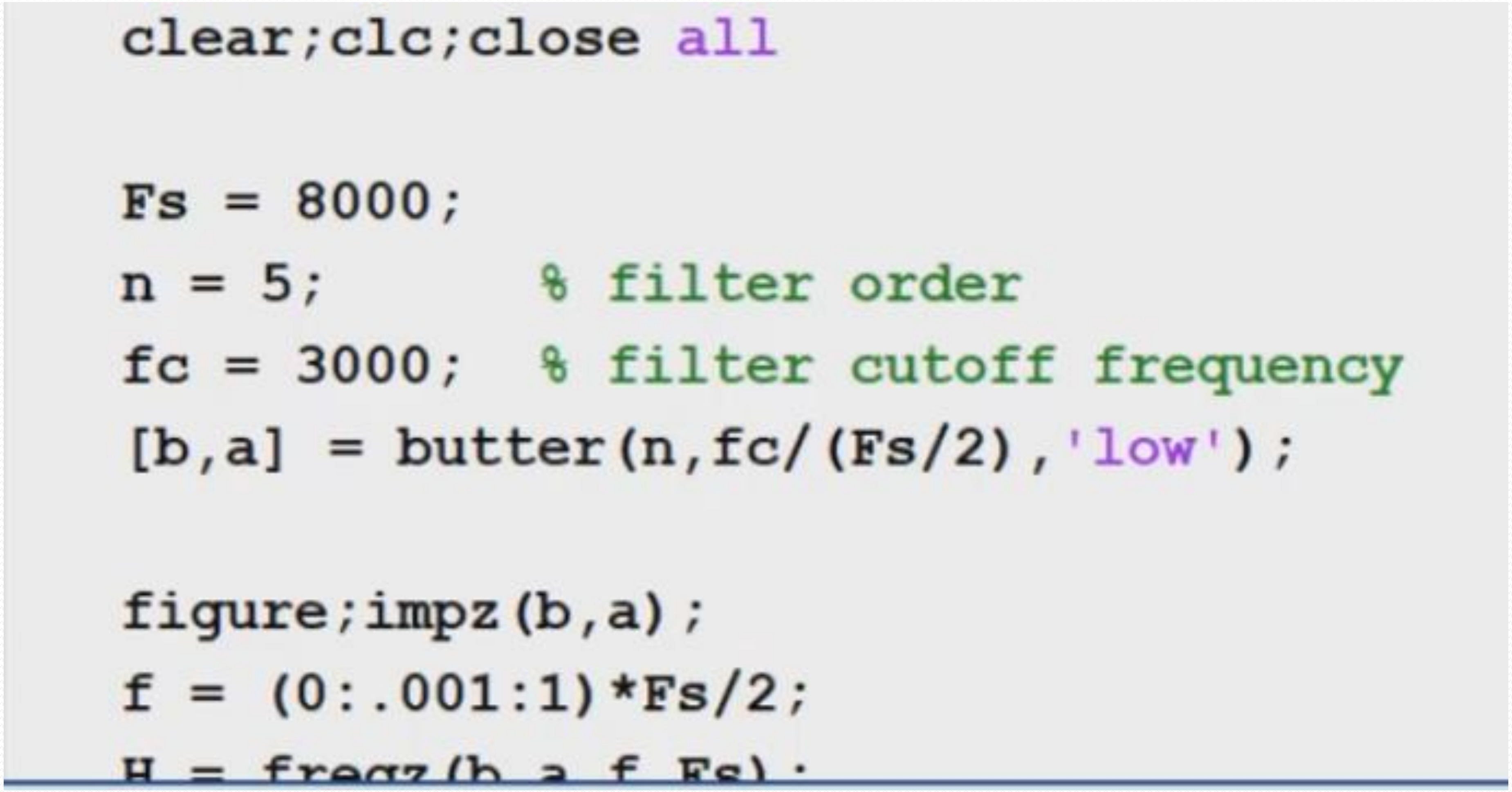
```
>> isstable(b,a)

ans =

  logical

   0
```

The ***Butterworth* filter** is one of the classic filter designs. It is characterized by having the "flattest" possible pass-band. Unfortunately, its performance in the stop band is decidedly mediocre.

```
n = 5;                    % filter order
fc = 1000;                % filter cutoff frequency
[b,a] = butter(n, fc/(Fs/2),'low');
```

After running MATLAB script we get the following equation:

$$y[n] = 0.2689 * x[n] + 1.3447 * x[n-1] + 2.6894 * x[n-2] + 2.6894 * x[n-3] + 1.3447 * x[n-4] + 0.2689 * x[n-5] - 2.4744 * y[n-1] - 2.811 * y[n-2] - 1.7038 * y[n-3] - 0.5444 * y[n-4] - 0.0723 * y[n-5]$$



```
clear;clc;close all

Fs = 8000;
n = 5;          % filter order
fc = 3000;      % filter cutoff frequency
[b,a] = butter(n,fc/(Fs/2),'low');

figure;impz(b,a);
f = (0:.001:1)*Fs/2;
H = freqz(b,a,f,Fs);
```

Filtering any signal using causal LTI system:

MATLAB has a function for performing this operation:

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

Then you can play the sound and plot its frequency spectrum.

Plotting frequency domain vs time domain: (spectrogram)

if you want to decide when a specific frequency appeared in the signal

```
specgram(x,512,Fs)
```