Assignment #2

1. Convolution

If arbitrary sequences are of infinite duration, then MATLAB cannot be used directly to compute the convolution. MATLAB does provide a built-in function called conv that computes the convolution between two finite-duration sequences. The `conv` function assumes that the two sequences begin at n = 0 and is invoked by

```
>> y = conv(x,h);
```

For example:

```
>> x = [3, 11, 7, 0, -1, 4, 2];  h = [2, 3, 0, -5, 2, 1];
>> y = conv(x, h)
y =
   6    31    47    6    -51   -5    41    18    -22    -3    8    2
```

However, the conv function neither provides nor accepts any timing information if the sequences have an arbitrary support. What is needed is a beginning point and an end point of `y(n)`. Given finite duration `x(n)` and `h(n)`, it is easy to determine these points.

Let

$\{x(n);\ n_{xb} \le n \le n_{xe}\}$ and $\{h(n);\ n_{hb} \le n \le n_{he}\}$

be two finite-duration sequences. Then referring to the example above we observe that the beginning and end points of y(n) are

$n_{yb} = n_{xb} + n_{hb}$ and $n_{ye} = n_{xe} + n_{he}$

respectively. A simple modification of the conv function, called `conv_m`, which performs the convolution of arbitrary support sequences can be designed

```
function [y,ny] = conv_m(x,nx,h,nh)
% Modified convolution routine for signal processing
% ---------------------------------------------
%  [y,ny] = conv_m(x,nx,h,nh)
%  [y,ny] = convolution result
%  [x,nx] = first signal
%  [h,nh] = second signal
%
nyb = nx(1)+nh(1); nye = nx(length(x)) + nh(length(h));
ny = [nyb:nye];   y = conv(x,h);
```

For signals *x[k]* and the impulse response *h[k]* below use Matlab to obtain the system response *y[k]*:

a)

$$x[k] = u[k] - u[k-8]$$
$$h[k] = \sin\left(\frac{2\pi k}{8}\right)(u[k] - u[k-8])$$

b)

$$x[k] = \sin\left(\frac{2\pi k}{8}\right)(u[k] - u[k-8])$$
$$h[k] = -\sin\left(\frac{2\pi k}{8}\right)(u[k] - u[k-8])$$

To do:

- Tape code to Matlab, run the code for case a and b.
- Plot figures presenting *x[k]*, impulse response *h[k]* and system response y*[k]* as functions of step number *k*.

TIPS: Use commands *`subplot(m,n,p)`* and *`stem`* to plot signals in one column under each other. Use commands *`xlabel, ylabel`* and *`title`* to label the plots.

2. Sampling

In this task, you should decimate a sampled signal, i.e. sample down the original signal to a signal with a lower sampling frequency. The decimation is done by reading every second value of the original signal. When sampling signals, the time between two successive readings is called the "sampling period" and its inverse "sampling frequency".

a) Create signal x

```
x = 4*cos(2*pi*0.1*k) + 2*cos(2*pi*0.35*k);
for k = 0:Ns-1; Ns = 5000;
```

It consists of two harmonic components with different frequencies.
Determine the signal's Fourier transform

```
h = fft(x);
```

Plot the variable *h*, use the scale corresponding to the sampling frequency *Fs = 1kHz* by creating a frequency axis f between 0 and Nyquist frequency

```
figure(1), plot(f,abs(h(1:Ns/2)))
```

Decimate signal *x* by creating a new vector `xd` that contains every other element of `x`

```
xd = x(1:2:end);
```

Determine spectrum of the signal *xd* with the command `fft` and present the result for the half samplings frequency *Fs = 500Hz* in `figure(2)` in the same way as above (note that the new Nyquist frequency and the number of samples have been halved and you should create a new frequency axis *fd)*.

Compare spectra in Figures 1 and 2 and draw conclusions

b) Repeat task a) for the signal

```
x = 4*cos(2*pi*0.1*k) + 2*cos(2*pi*0.4*k);
för k=1:Ns; Ns = 5000;
```

Note the change in the spectrum in `figure(2)`.

To do:

- Plot spectra in figures 1 and 2 for the cases a) and b).
- Discuss results: compare different spectra and comment on the differences. What frequencies are obtained after decimation and why?