06.05.2025

AGH, Signals and Systems Krzysztof Smykla

Lab Assignment 2
Report

Problem 1 - Convolution:

In problem 1, the task was to calculate the convolution of signals *x[k]* and the impulse response *h[k]* to obtain the system response *y[k]* of the following signals:

a)

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

b)

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

Fig. 1 – Input signal x[k] and impulse response h[k]

The convolution operation is defined by the following equation:

$$\boxed{y(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k)} \quad (1)$$

Since the operation is carried out in **discrete time** the formula uses the summation operator.

To perform the convolution of signals x[k] and h[k], I used the modified convolution function *conv_m()* in MATLAB. The function *conv_m()* is defined by the following script:

SCRIPT 1:

```
function [y, ny] = conv_m(x, nx, h, nh)

% Convolution routine for signal processing

% ------------------------------------------

% [y, ny] = conv_m(x, nx, h, nh)

% [y, ny] = convolution result and its time index

% [x, nx] = first signal and its time indices

% [h, nh] = second signal and its time indices



% Determine the starting and ending indices of the output
```

```
nyb = nx(1) + nh(1);                  % Start of output time index

nye = nx(end) + nh(end);             % End of output time index

ny = nyb:nye;                         % Output time index vector



% Perform the convolution

y = conv(x, h);

end
```

a)

In point a) the convolved signals were:

x[k] = $u[k] - u[k-8]$

h[k] = $sin(2\pi k/8)(u[k] - u[k-8])$

where u is the unit step function.

Using the modified convolution function conv_m() I calculated the convolution in MATLAB using the following script:

SCRIPT 2:

```
 % Defining a unit step function u

% Start with all zeros:

k = -10:10;

u = @(k) double(k >= 0);



% Point a)

x = u(k) - u(k - 8);

h = sin(2*pi*k/8) .* (u(k) - u(k - 8));



figure(1);

% Input Signal

subplot(2,1,1);

stem(k, x, 'filled', 'LineWidth', 2);

title('x[k]'); xlabel('k'); ylabel('x[k]');

% Impulse Response

subplot(2,1,2);
```

```
stem(k, h, 'filled', 'LineWidth', 2);

title('h[k]'); xlabel('k'); ylabel('h[k]');


% Convolution of x(k) * h(k) using modified conv_m() function

y = conv_m(x,k,h,k)


figure(2);

plot(y,"-b","LineWidth",2)

title('y = x[k] * h[k]'); xlabel('k'); ylabel('y[k]')
```

Explanation:

The MATLAB code above generates a discrete-valued plot of the input signal x[k] and the impulse response h[k]. It then computes the convolution using the *conv_m()* function defined in script 1. Finally, the script generates a plot of the resulting operation.

The images are presented in the following figures:

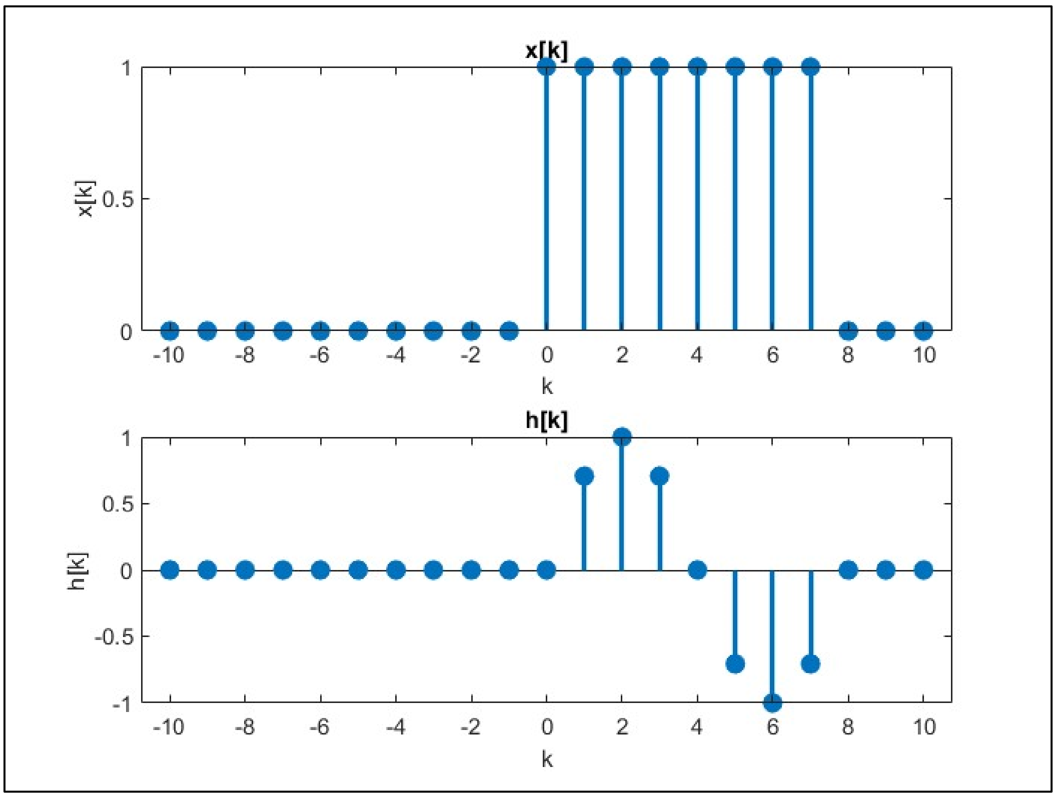


Fig. 2 - Plots of signals x[k] and h[k]

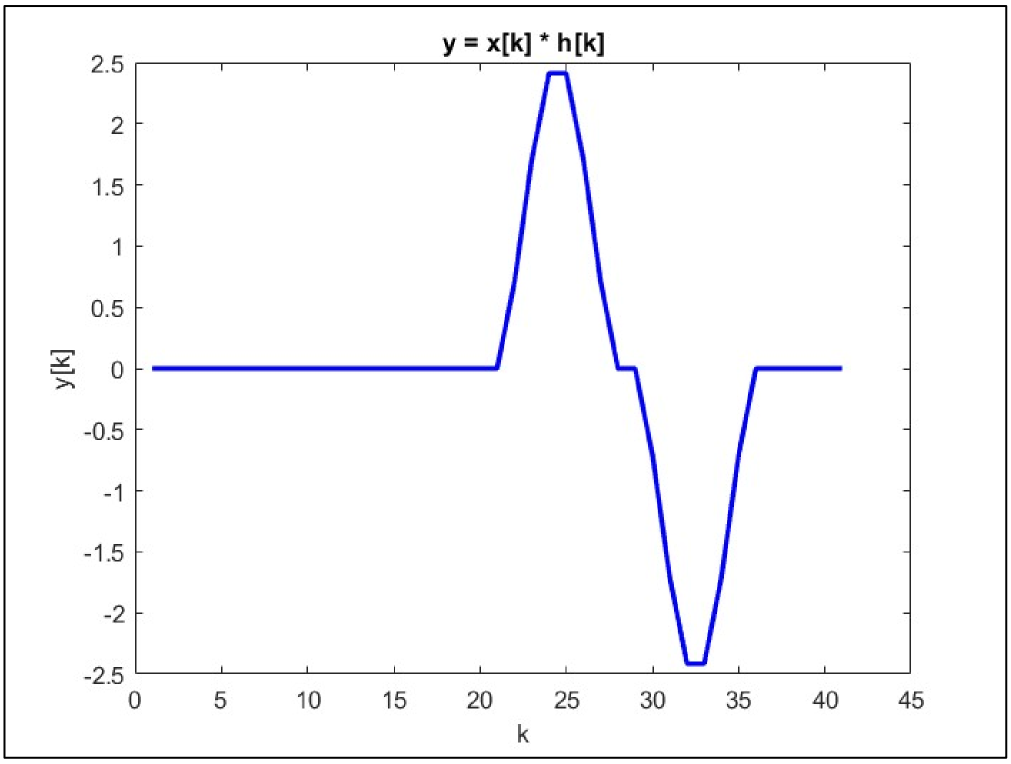


Fig. 3 – Convolution y[k] = x[k]*h[k]

b)

In point b) the convolved signals were:

x2[k] = $sin(2\pi k/8)(u[k] - u[k-8])$

h2[k] = $-\ sin(2\pi k/8)(u[k] - u[k-8])$

The convolution is analogous to point a)

The generated images are shown below:

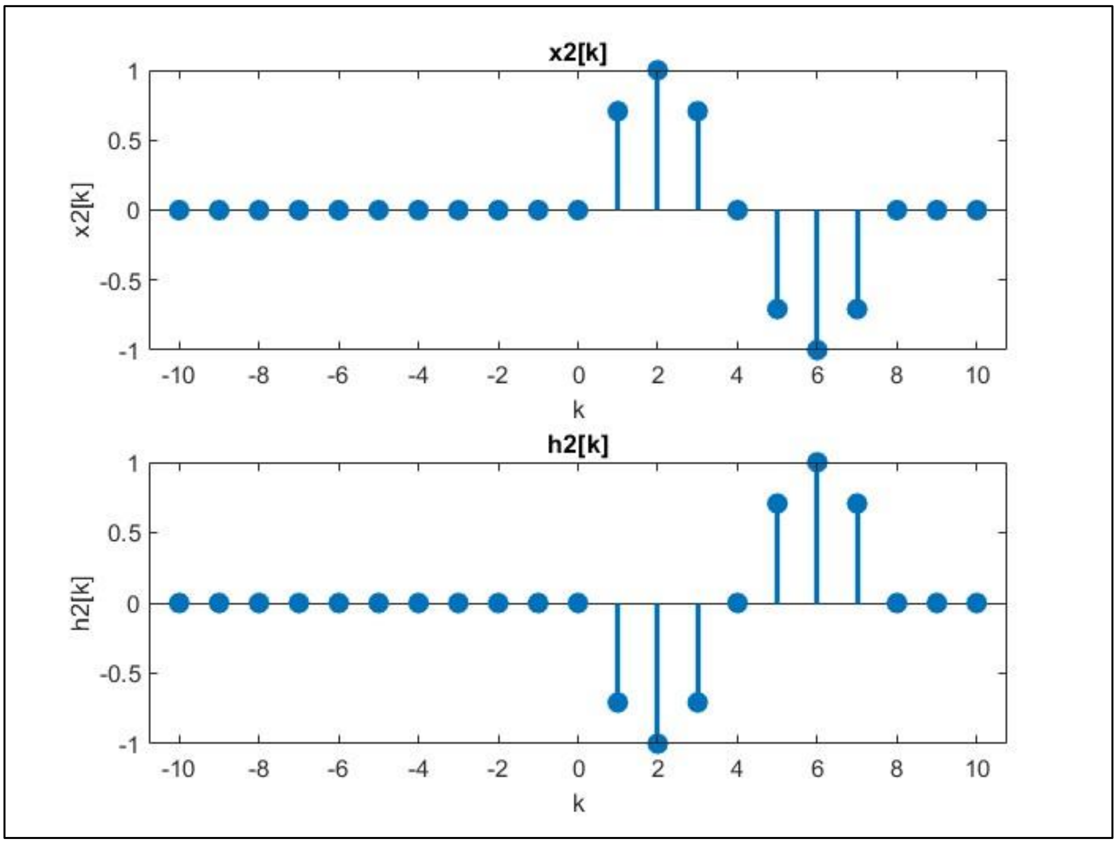


Fig. 4 – Plots of signals x2[k] and h2[k]

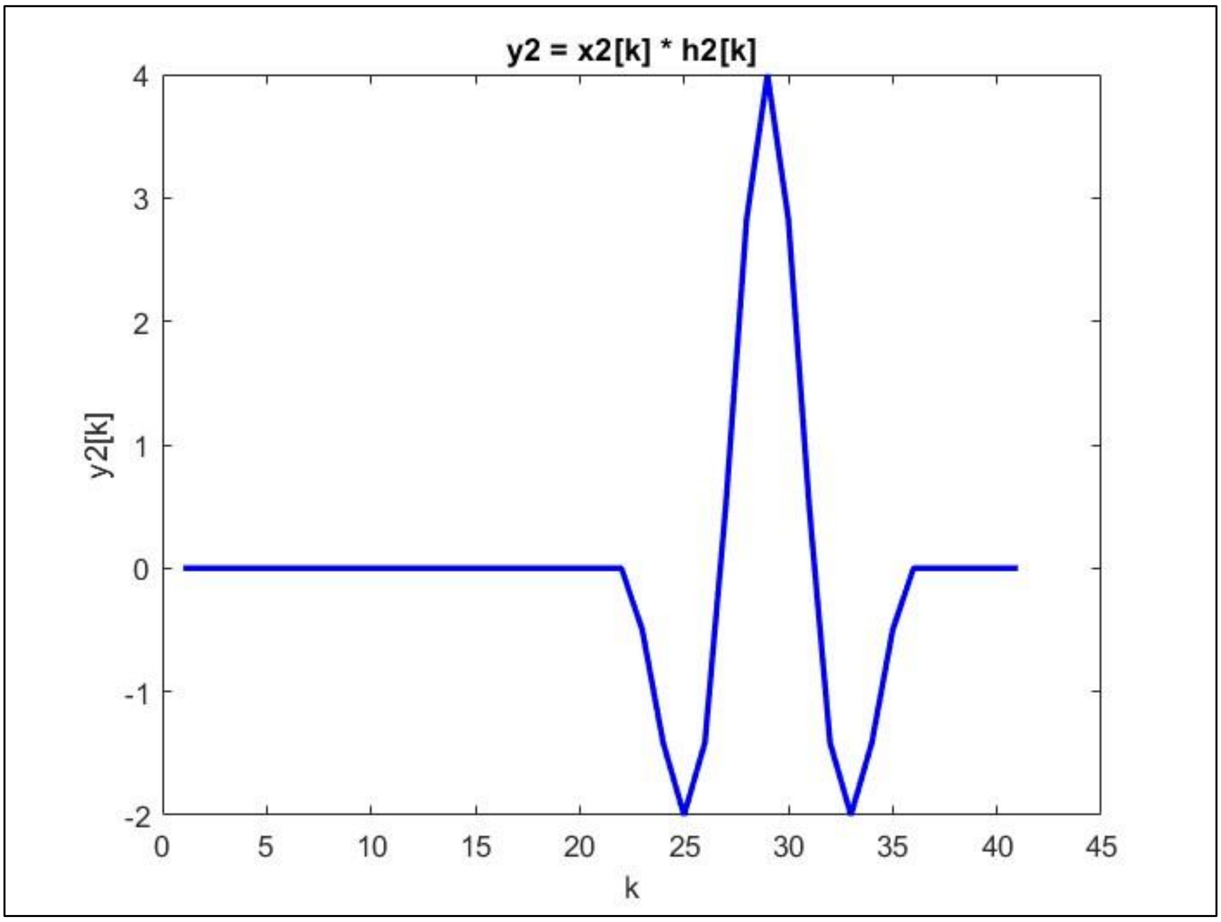


Fig. 5 – Convolution y2[k] = x2[k]*h2[k]

Below I have provided the full MATLAB code which generates all of the plots shown above for points a) and b).

SCRIPT 3:

```
% Assignment #2


% 1. Convolution


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


% Defining a unit step function u

% Start with all zeros:

k = -10:10;

u = @(k) double(k >= 0);


% Point a)

x = u(k) - u(k - 8);

h = sin(2*pi*k/8) .* (u(k) - u(k - 8));


figure(1);

% Input Signal

subplot(2,1,1);

stem(k, x, 'filled', 'LineWidth', 2);

title('x[k]'); xlabel('k'); ylabel('x[k]');

% Impulse Response

subplot(2,1,2);

stem(k, h, 'filled', 'LineWidth', 2);

title('h[k]'); xlabel('k'); ylabel('h[k]');


% Convolution of x(k) * h(k) using modified conv_m() function

y = conv_m(x,k,h,k)


figure(2);

plot(y,"-b","LineWidth",2)
```

```
title('y = x[k] * h[k]'); xlabel('k'); ylabel('y[k]')



% Point b)

x2 = sin(2*pi*k/8) .* (u(k) - u(k - 8));

h2 = -sin(2*pi*k/8) .* (u(k) - u(k - 8));



figure(3);

% Input Signal

subplot(2,1,1);

stem(k, x2, 'filled', 'LineWidth', 2);

title('x2[k]'); xlabel('k'); ylabel('x2[k]');

% Impulse Response

subplot(2,1,2);

stem(k, h2, 'filled', 'LineWidth', 2);

title('h2[k]'); xlabel('k'); ylabel('h2[k]');

% Convolution of x2(k) * h2(k)using modified conv_m() function

y2 = conv_m(x2,k,h2,k)



figure(4);

plot(y2,"-b","LineWidth",2)

title('y2 = x2[k] * h2[k]'); xlabel('k'); ylabel('y2[k]')
```

SUMMARY:

In Problem 1, the task was to compute the convolution of discrete-time signals using MATLAB. A custom function conv_m() was implemented to calculate the convolution and return both the result and the corresponding time indices. Two sets of signals were analyzed. In **Part a)**, a rectangular pulse x[k]x[k]x[k] was convolved with a modulated sinusoidal pulse h[k]h[k]h[k], both defined using unit step functions. The resulting output y[k]y[k]y[k] illustrated the system's response over time. In **Part b)**, two modulated sine waves of equal length but opposite signs were convolved, demonstrating how opposing phase components interact during convolution. MATLAB plots confirmed the shapes and durations of the resulting signals.

Problem 2 - Sampling:

Problem 2 asks us to sample two signals at different frequencies, the Nyquist frequency and a frequency exceeding it.

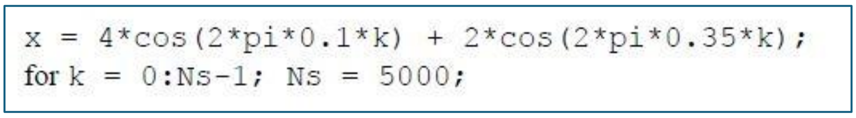
The Nyquist frequency – being the highest frequency that equipment of a given sample rate can reliably measure, **one-half the given sample rate**.

If the signal's highest frequency component exceeds the Nyquist frequency, it can lead to aliasing, where high-frequency information is misinterpreted as lower frequencies.

A higher sampling rate than the Nyquist frequency, causes the signal to become decimated.

a)

In point a) we are given the following signal:



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

Fig. 6 – Signal X

The sampling index k ranges from 0 to Ns – 1. The number of samples Ns = 5000.

The code below generates the plots of signal x in the frequency domain after performing the FFT.

SCRIPT 4:

```
 % 2. Sampling



clear

clearvars



% Signal a)

Ns = 5000;

%k = 1:Ns;

k = 0:Ns-1;

% Signal x

x = 4*cos(2*pi*0.1*k) +2*cos(2*pi*0.35*k);



figure(1)

% The fourier transform of x

h = fft(x);

f= linspace(0,1000,Ns);

subplot(2,1,1)
```

```
% Plotting the signel between 0 and the Nyquist frequency Ns

stem(f,abs(h(1:Ns))/Ns), grid

title('fs = 1000Hz')

xlabel('Frequency in Hz')

ylabel('DFT Amplitude')



subplot(2,1,2)

stem(f,real(h)/Ns)

hold on

stem(f,imag(h)/Ns)

hold off

title('fs = 1000Hz')

xlabel('Frequency in Hz')

ylabel('Amplitude')

legend('Real H','Imag H')

grid
```

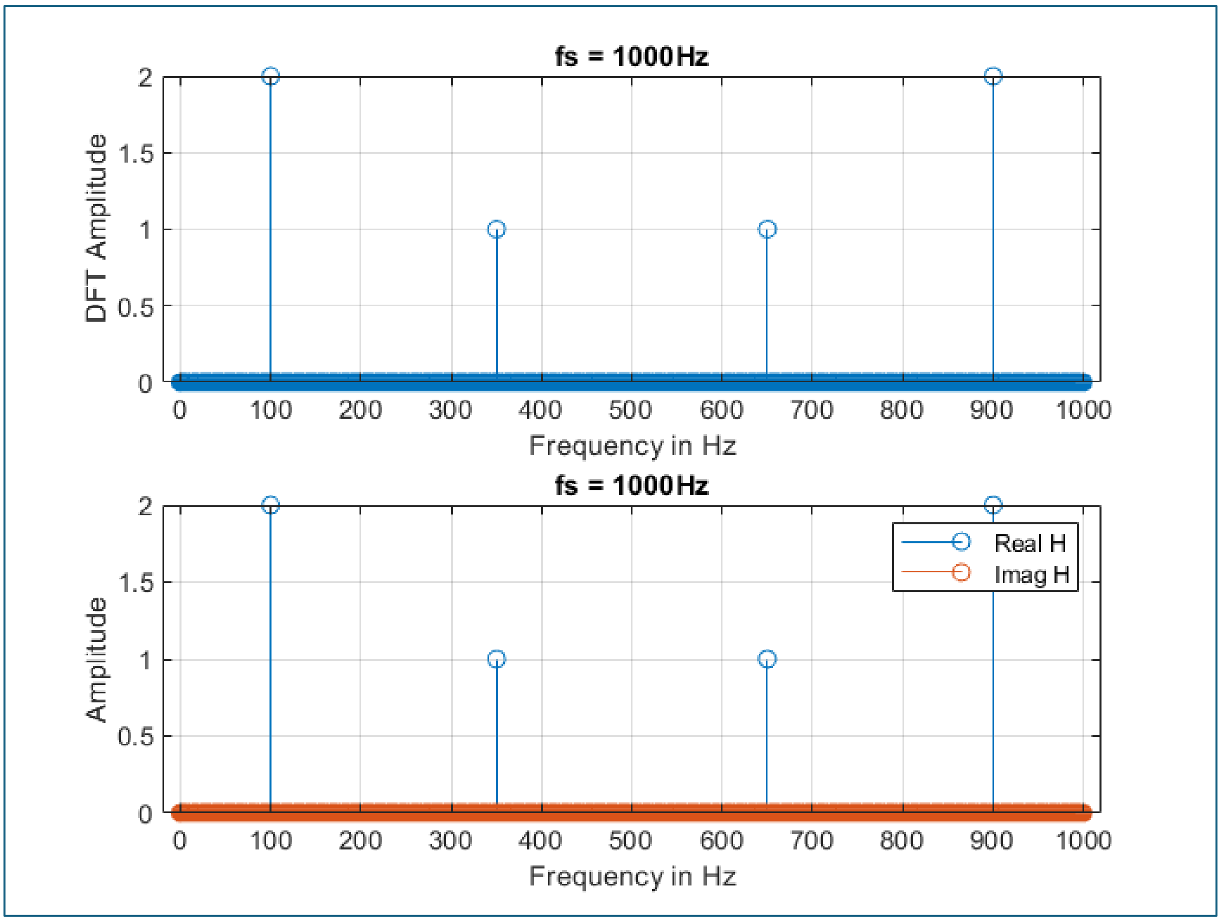


Fig. 7 – Signal X, FFT

Then, signal x is decimated- that is sampled at a lower frequency than the original signal.

SCRIPT 5:

```
 % Decimation of signal x

xd = x(1:2:end); % Decimate signal

Nsd = length(xd);

hd = fft(xd);

fd = linspace(0,500,Nsd);

% Plotting decimated signal

figure(2)

subplot(211)

stem(fd,abs(hd(1:Nsd))/Nsd), grid

title('fs = 500Hz')

xlabel('Frequency in Hz')

ylabel('DFT Amplitude')

subplot(212)

stem(fd,real(hd)/Nsd)

hold on

stem(fd,imag(hd)/Nsd)

hold off

title('fs = 500Hz')

xlabel('Frequency in Hz')

ylabel('Aamplitude')

legend('Real H','Imag H')

grid
```

The plots below show the decimated signal after sampling at a lower sampling rate:

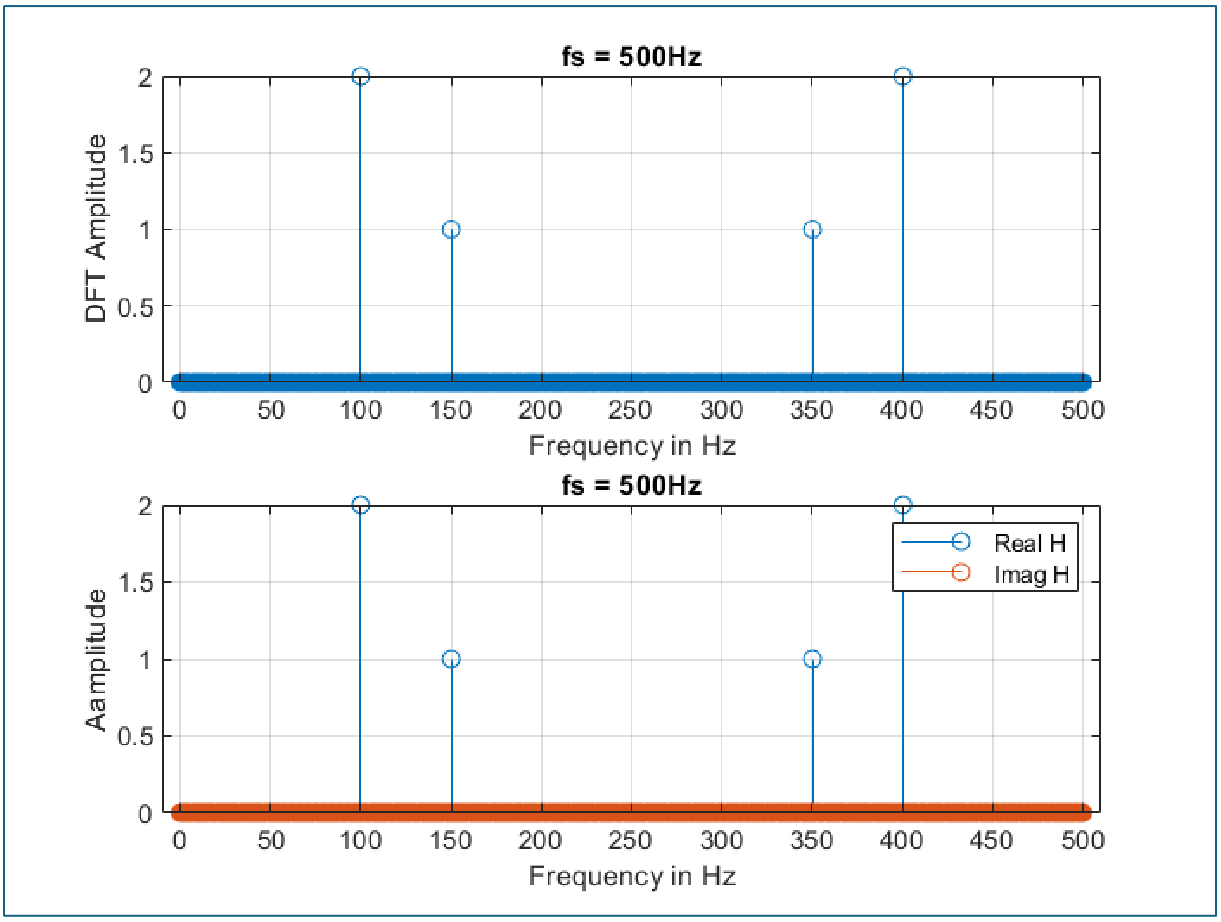
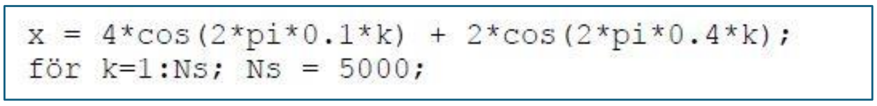


Fig. 8 – decimated signal x

The difference between plots shown on figures 7 and 8 is the frequency values present in the signal. The decimated signal (fig. 8) shows an amplitude of 1 at frequency 150 hz which is lower than that of the original signal (350 hz). This means that the higher frequencies of the original signal are being misrepresented as lower ones.

b)

In point b) the given signal was:



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

Fig. 9 – Signal X

The sample index k ranges from 1 to Ns.

The script below generates the corresponding plots as in the case of point a.

SCRIPT 6:

```
 % Signal b)

Ns = 5000;

k = 1:Ns;

%k = 0:Ns-1;

x = 4*cos(2*pi*0.1*k) +2*cos(2*pi*0.4*k);

figure(1)
```

```
h = fft(x);

f= linspace(0,1000,Ns);

subplot(211)

stem(f,abs(h(1:Ns))/Ns), grid

title('fs = 1000Hz')

xlabel('Frequency in Hz')

ylabel('DFT Amplitude')

subplot(212)

stem(f,real(h)/Ns)

hold on

stem(f,imag(h)/Ns)

hold off

title('fs = 1000Hz')

xlabel('Frequency in Hz')

ylabel('Amplitude')

legend('Real H','Imag H')

grid
```

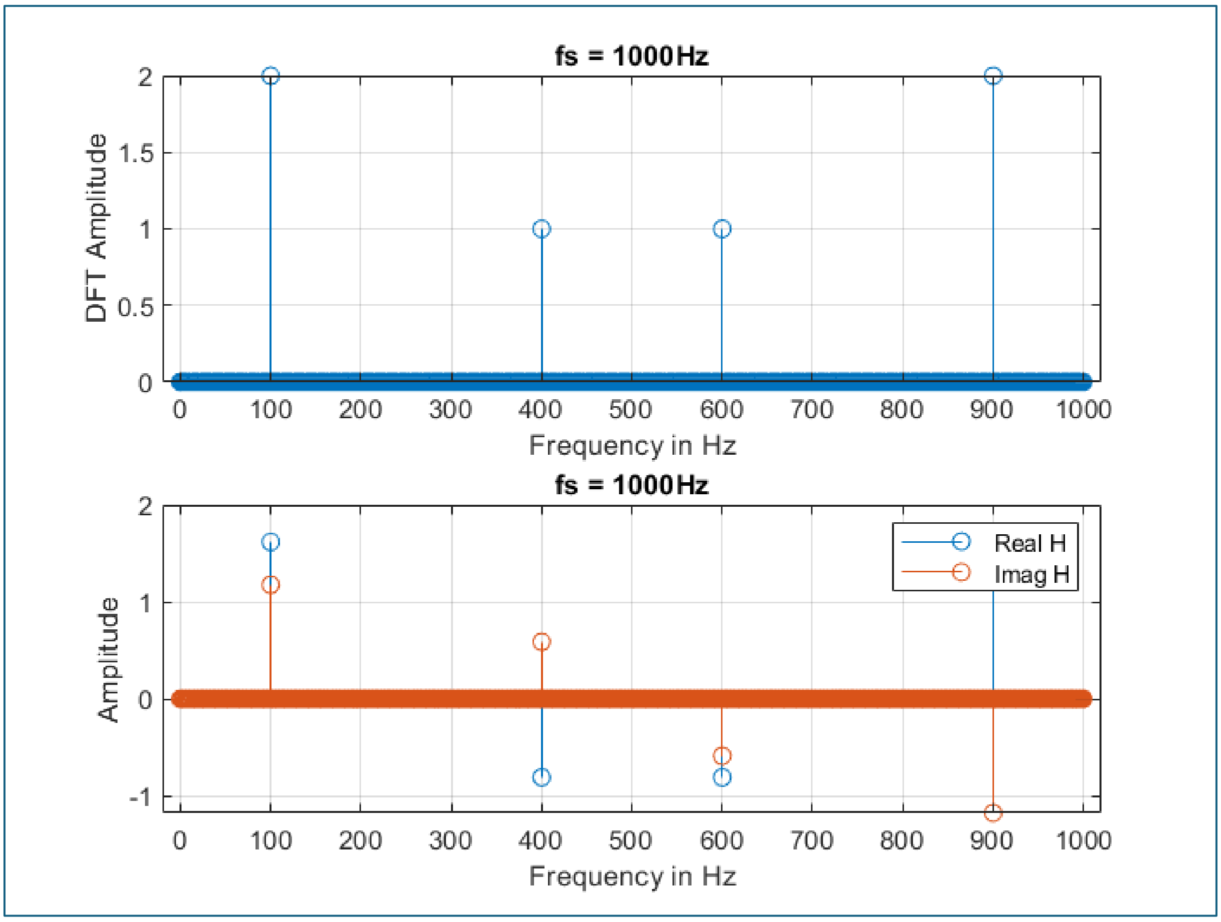


Fig. 9 – Signal x, pt. b

The signal is later decimated, analogously to point a).

SCRIPT 7:

```
 % Decimation

xd = x(1:2:end); % Decimate signal

Nsd = length(xd);

hd = fft(xd);

fd = linspace(0,500,Nsd);



figure(2)

subplot(211)

stem(fd,abs(hd(1:Nsd))/Nsd), grid

title('fs = 500Hz')

xlabel('Frequency in Hz')

ylabel('DFT Amplitude')

subplot(212)

stem(fd,real(hd)/Nsd)

hold on

stem(fd,imag(hd)/Nsd)

hold off

title('fs = 500Hz')

xlabel('Frequency in Hz')

ylabel('Aamplitude')

legend('Real H','Imag H')

grid
```

The decimated signal is shown on the plot below:

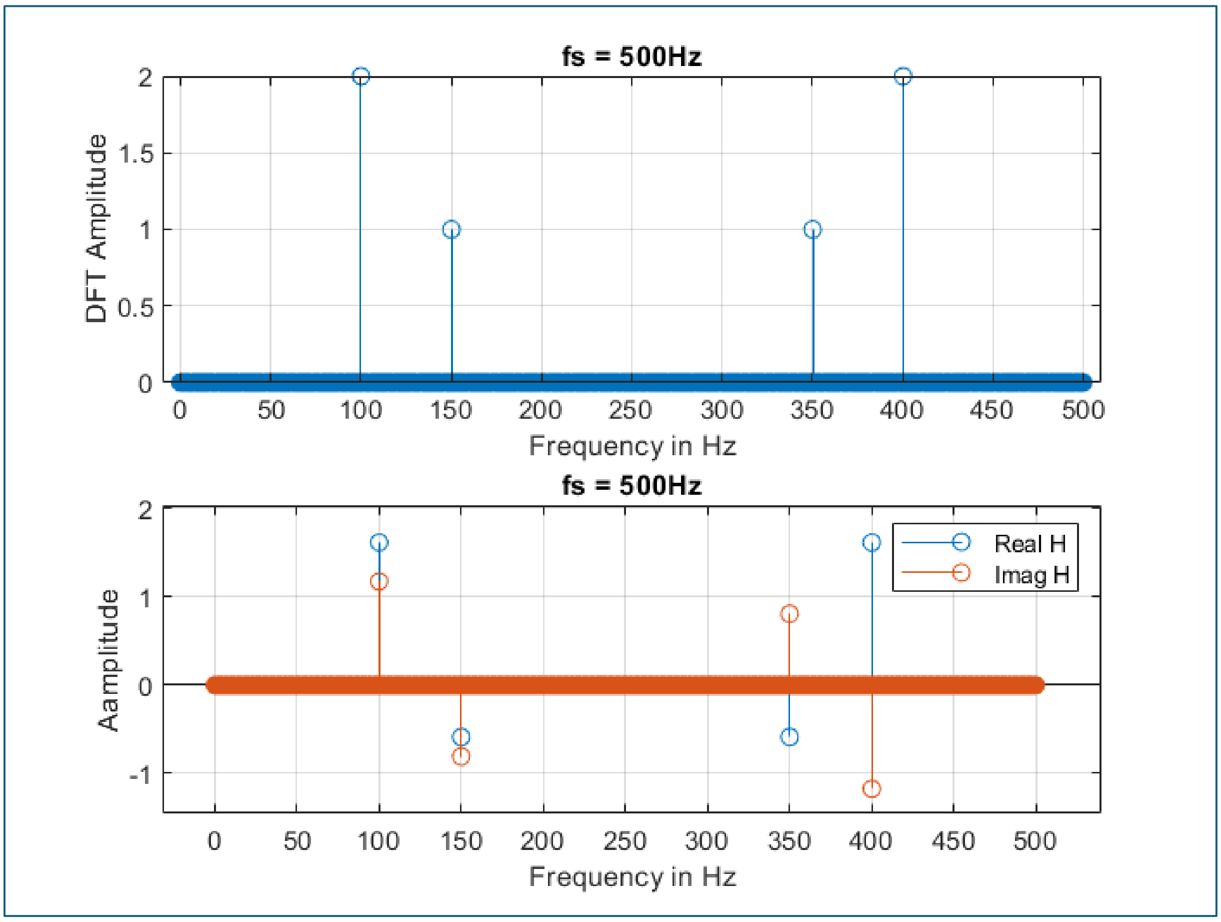


Fig. 10 – Decimated signal x, pt. b

Similarly to Point a, in Point b the decimated signal misrepresents frequency components above the Nyquist frequency (which is half the sampling rate). This results in aliasing, where higher frequency components are reflected as lower frequencies in the spectrum. As a result, the frequency content of the signal is distorted in the decimated version and does not accurately represent the original signal.

SUMMARY:

Problem 2 explored the effects of sampling a composite signal at and below the Nyquist rate. In **Part a)**, a signal consisting of cosine waves at 0.1 and 0.35 normalized frequency was sampled at 1000 Hz (above Nyquist) and then decimated to 500 Hz (below Nyquist). The FFT of the decimated signal revealed **aliasing**, where the 350 Hz component was incorrectly shifted to a lower frequency. In **Part b)**, a similar test was run using a component at 0.4 normalized frequency. Again, aliasing was observed in the decimated version, confirming that undersampling leads to distortion in frequency representation. The experiments emphasized the importance of meeting the Nyquist criterion to avoid frequency misinterpretation.

FULL MATLAB CODE SOLUTION:

```
 % Assignment #2

% 1. Convolution

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

% Defining a unit step function u
% Start with all zeros:
k = -10:10;
u = @(k) double(k >= 0);

% Point a)
x = u(k) - u(k - 8);
h = sin(2*pi*k/8) .* (u(k) - u(k - 8));

figure(1);
% Input Signal
subplot(2,1,1);
stem(k, x, 'filled', 'LineWidth', 2);
title('x[k]'); xlabel('k'); ylabel('x[k]');
% Impulse Response
subplot(2,1,2);
stem(k, h, 'filled', 'LineWidth', 2);
title('h[k]'); xlabel('k'); ylabel('h[k]');

% Convolution of x(k) * h(k) using modified conv_m() function
y = conv_m(x,k,h,k)

figure(2);
plot(y,"-b","LineWidth",2)
title('y = x[k] * h[k]'); xlabel('k'); ylabel('y[k]')
```

```
% Point b)

x2 = sin(2*pi*k/8) .* (u(k) - u(k - 8));

h2 = -sin(2*pi*k/8) .* (u(k) - u(k - 8));


figure(3);

% Input Signal

subplot(2,1,1);

stem(k, x2, 'filled', 'LineWidth', 2);

title('x2[k]'); xlabel('k'); ylabel('x2[k]');

% Impulse Response

subplot(2,1,2);

stem(k, h2, 'filled', 'LineWidth', 2);

title('h2[k]'); xlabel('k'); ylabel('h2[k]');

% Convolution of x2(k) * h2(k)using modified conv_m() function

y2 = conv_m(x2,k,h2,k)


figure(4);

plot(y2,"-b","LineWidth",2)

title('y2 = x2[k] * h2[k]'); xlabel('k'); ylabel('y2[k]')


%%

% 2. Sampling


clear

clearvars


% Signal a)

Ns = 5000;

%k = 1:Ns;

k = 0:Ns-1;

% Signal x

x = 4*cos(2*pi*0.1*k) +2*cos(2*pi*0.35*k);
```

```
figure(1)

% The fourier transform of x

h = fft(x);

f= linspace(0,1000,Ns);

subplot(2,1,1)

% Plotting the signel between 0 and the Nyquist frequency Ns

stem(f,abs(h(1:Ns))/Ns), grid

title('fs = 1000Hz')

xlabel('Frequency in Hz')

ylabel('DFT Amplitude')


subplot(2,1,2)

stem(f,real(h)/Ns)

hold on

stem(f,imag(h)/Ns)

hold off

title('fs = 1000Hz')

xlabel('Frequency in Hz')

ylabel('Amplitude')

legend('Real H','Imag H')

grid

%%

% Decimation of signal x

xd = x(1:2:end); % Decimate signal

Nsd = length(xd);

hd = fft(xd);

fd = linspace(0,500,Nsd);

% Plotting decimated signal

figure(2)

subplot(211)

stem(fd,abs(hd(1:Nsd))/Nsd), grid

title('fs = 500Hz')

xlabel('Frequency in Hz')
```

```
ylabel('DFT Amplitude')

subplot(212)

stem(fd,real(hd)/Nsd)

hold on

stem(fd,imag(hd)/Nsd)

hold off

title('fs = 500Hz')

xlabel('Frequency in Hz')

ylabel('Aamplitude')

legend('Real H','Imag H')

grid

%%

% Signal b)

Ns = 5000;

k = 1:Ns;

%k = 0:Ns-1;

x = 4*cos(2*pi*0.1*k) +2*cos(2*pi*0.4*k);


figure(1)

h = fft(x);

f= linspace(0,1000,Ns);

subplot(211)

stem(f,abs(h(1:Ns))/Ns), grid

title('fs = 1000Hz')

xlabel('Frequency in Hz')

ylabel('DFT Amplitude')

subplot(212)

stem(f,real(h)/Ns)

hold on

stem(f,imag(h)/Ns)

hold off

title('fs = 1000Hz')

xlabel('Frequency in Hz')
```

```
ylabel('Amplitude')

legend('Real H','Imag H')

grid



% Decimation

xd = x(1:2:end); % Decimate signal

Nsd = length(xd);

hd = fft(xd);

fd = linspace(0,500,Nsd);



figure(2)

subplot(211)

stem(fd,abs(hd(1:Nsd))/Nsd), grid

title('fs = 500Hz')

xlabel('Frequency in Hz')

ylabel('DFT Amplitude')

subplot(212)

stem(fd,real(hd)/Nsd)

hold on

stem(fd,imag(hd)/Nsd)

hold off

title('fs = 500Hz')

xlabel('Frequency in Hz')

ylabel('Aamplitude')

legend('Real H','Imag H')

grid
```