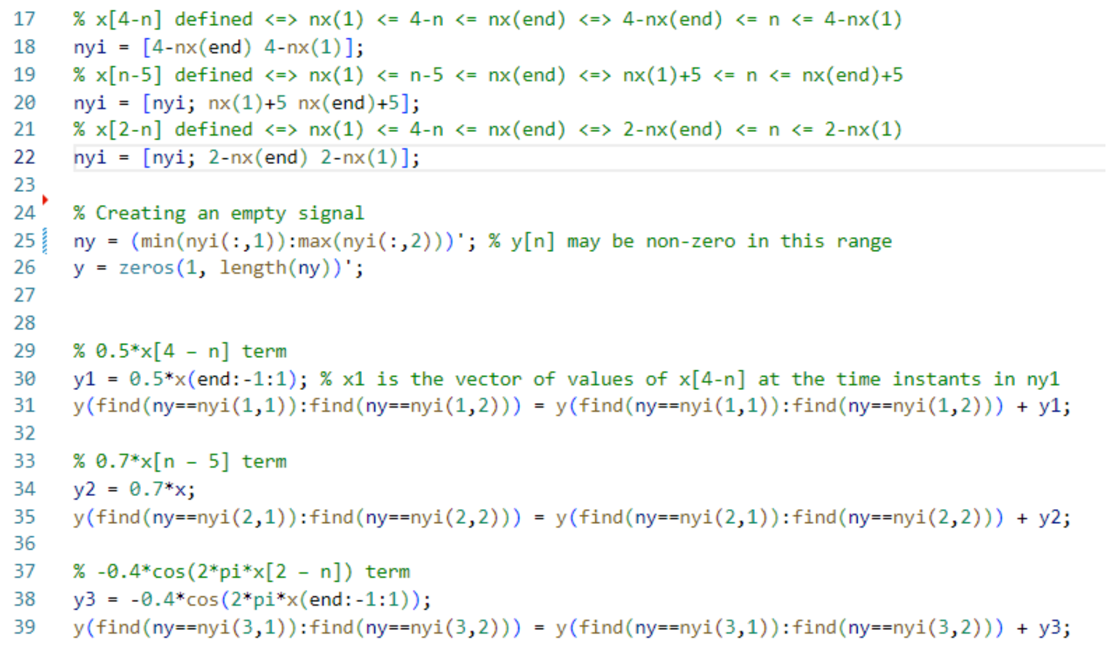
Systems Lab: Report

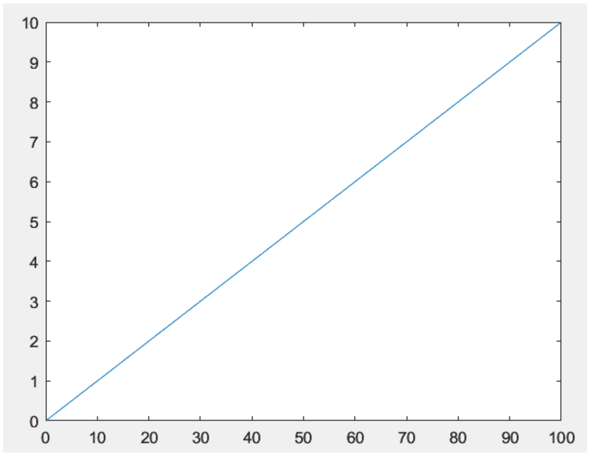
Project 1: Implement discrete-time systems as functions in Matlab

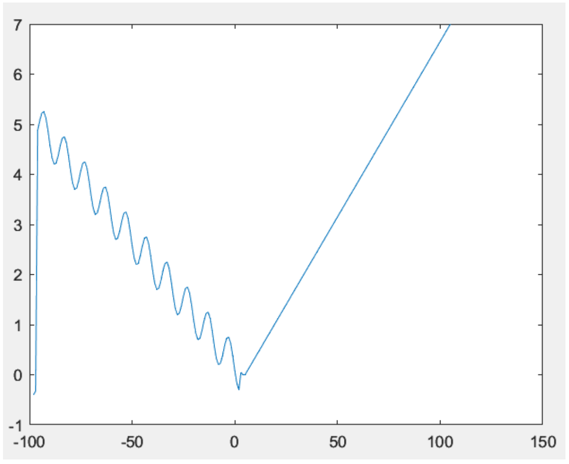
To create this function, I first defined the time ranges in which each term of the system would be acting, by solving for n in the boundary inequalities. Then, I created an empty output signal, with time values covering all of the previous ranges. Then, I calculated the values of each term, and added them to the corresponding section in the output signal vector.



```
% x[4-n] defined <=> nx(1) <= 4-n <= nx(end) <=> 4-nx(end) <= n <= 4-nx(1)
nyi = [4-nx(end) 4-nx(1)];
% x[n-5] defined <=> nx(1) <= n-5 <= nx(end) <=> nx(1)+5 <= n <= nx(end)+5
nyi = [nyi; nx(1)+5 nx(end)+5];
% x[2-n] defined <=> nx(1) <= 4-n <= nx(end) <=> 2-nx(end) <= n <= 2-nx(1)
nyi = [nyi; 2-nx(end) 2-nx(1)];

% Creating an empty signal
ny = (min(nyi(:,1)):max(nyi(:,2)))'; % y[n] may be non-zero in this range
y = zeros(1, length(ny))';


% 0.5*x[4 - n] term
y1 = 0.5*x(end:-1:1); % x1 is the vector of values of x[4-n] at the time instants in ny1
y(find(ny==nyi(1,1)):find(ny==nyi(1,2))) = y(find(ny==nyi(1,1)):find(ny==nyi(1,2))) + y1;

% 0.7*x[n - 5] term
y2 = 0.7*x;
y(find(ny==nyi(2,1)):find(ny==nyi(2,2))) = y(find(ny==nyi(2,1)):find(ny==nyi(2,2))) + y2;

% -0.4*cos(2*pi*x[2 - n]) term
y3 = -0.4*cos(2*pi*x(end:-1:1));
y(find(ny==nyi(3,1)):find(ny==nyi(3,2))) = y(find(ny==nyi(3,1)):find(ny==nyi(3,2))) + y3;
```

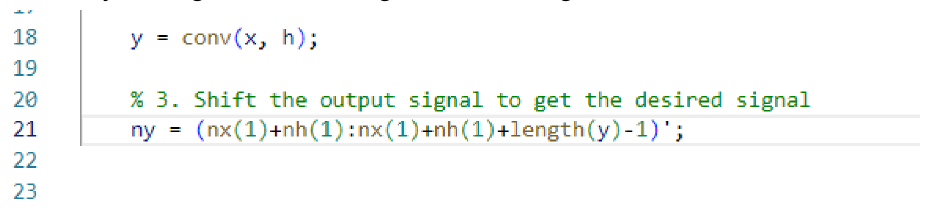
I did a test using the signal on the left, which got me the signal on the right as an output.





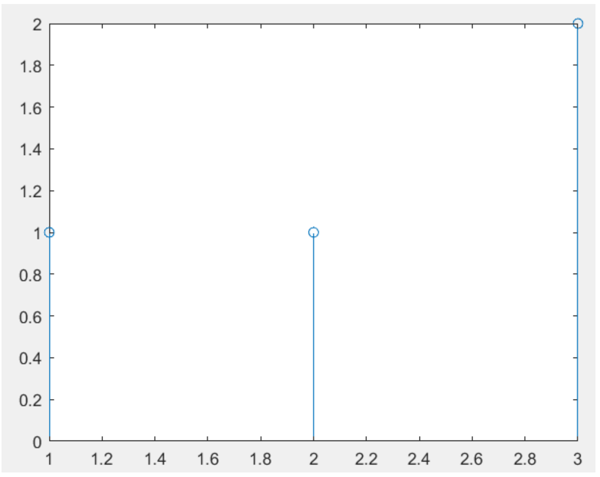
Project 2: Implementing the discrete-time convolution between finite-length signals

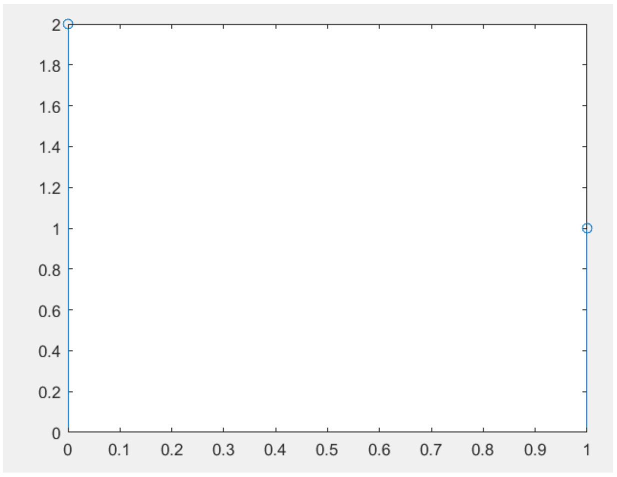
By using these two lines, we can reuse the `conv(x,h)` function and then shift the time interval vector by the right amount to get the final signal.

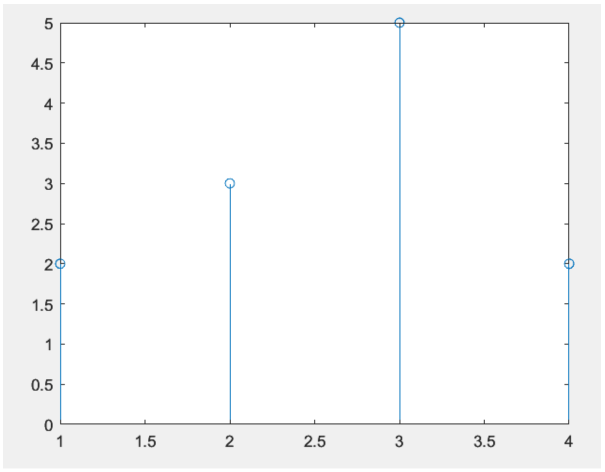


```
    y = conv(x, h);

    % 3. Shift the output signal to get the desired signal
    ny = (nx(1)+nh(1):nx(1)+nh(1)+length(y)-1)';


```

I did a test using the two signals above, and got as a result the signal below.

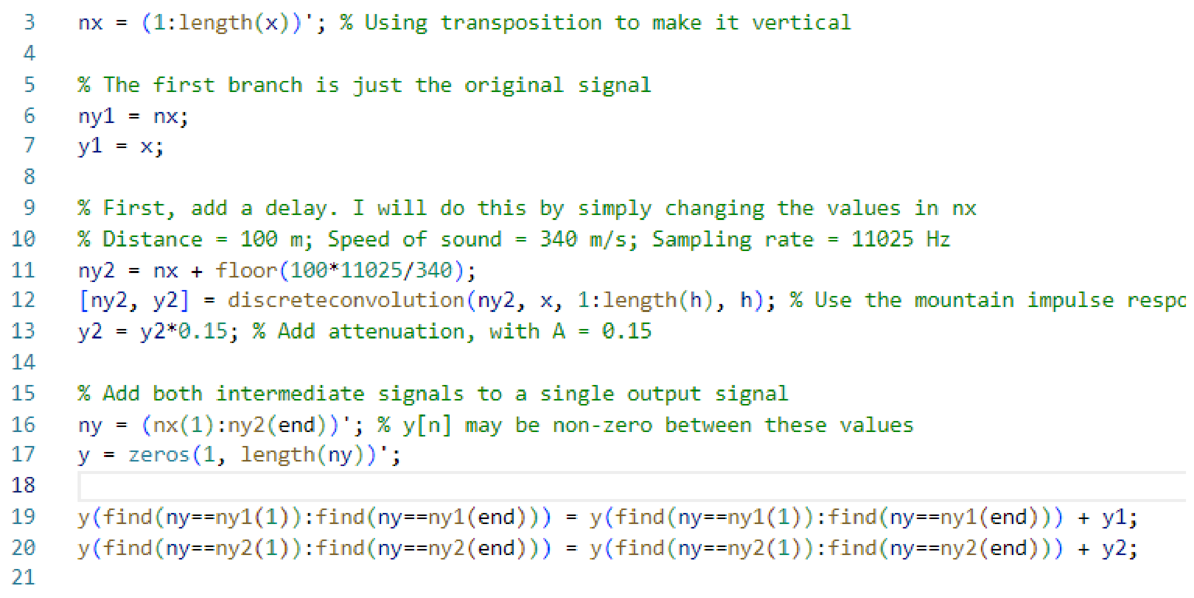






Project 3: Simulating an audio signal with echo

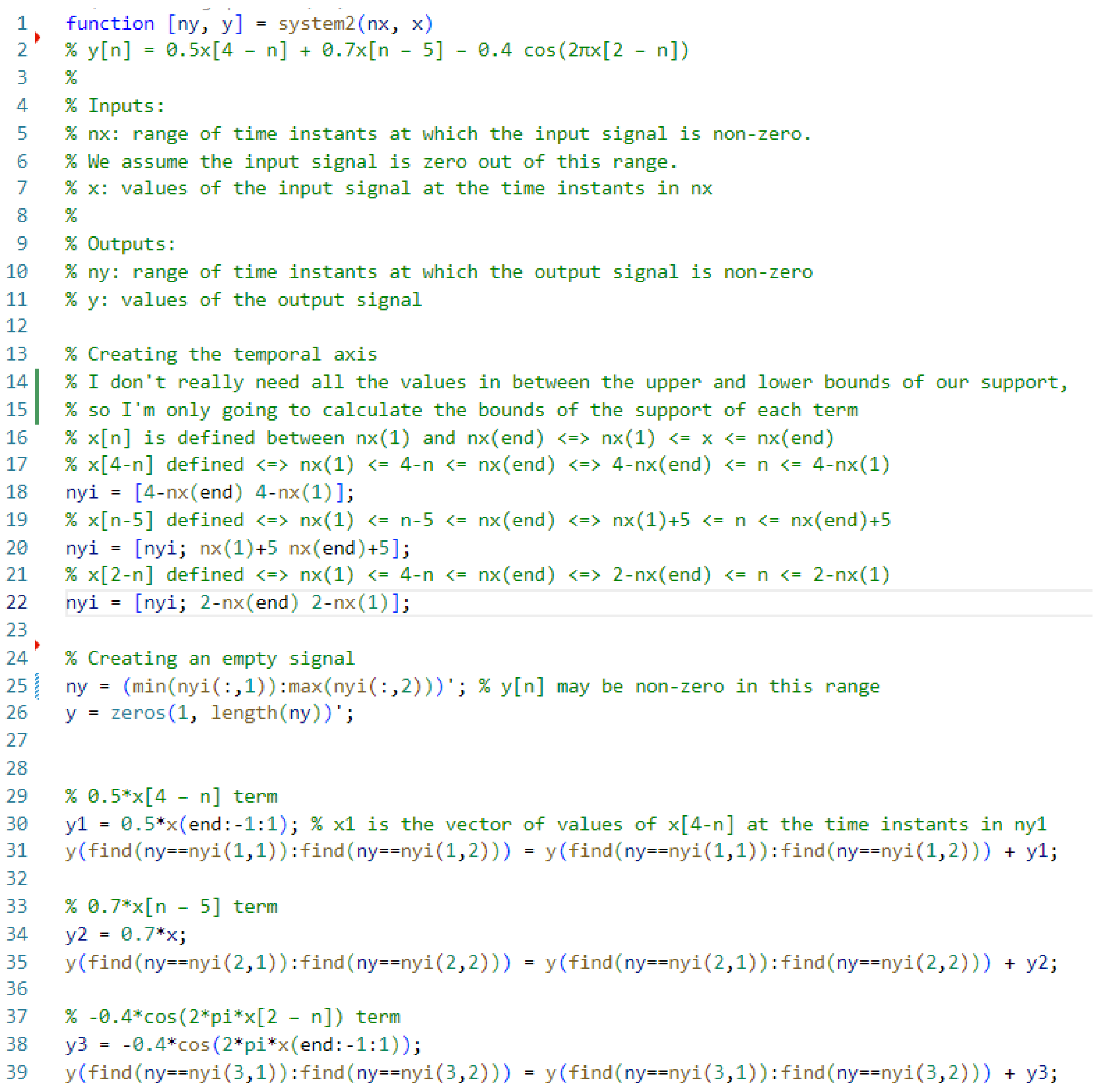
I used this code to add echo to the signal x (obtained using `audioread`). This is going to be the sum of two terms, the first being the original audio, and the second being a distorted version with a delay. It is not really necessary, but for the sake of clearneess, I created each of these terms in a separate signal.
The first signal is `y1`, a clone of the original sound signal. The second one is `y2`, created by adding a constant to every item in the time point vector (which will result in a delay), calculating the convolution with a given impulse response signal `h`, and adding some attenuation. Then, the final signal is created and initialized to 0, in the time range between the original start time and until the original end time plus the echo delay. Finally, both terms are added.
This delay is supposed to correspond to a distance of 100 m: at 340 m/s, it should take a sound wave 0.29 seconds. Since the sample rate is 11025, this corresponds to approximately 3242 samples (rounded so the sample instants align with the other term). The attenuation in my case was decided to be 0.15. These values can be tweaked, but they sounded good to me.



```
nx = (1:length(x))'; % Using transposition to make it vertical

% The first branch is just the original signal
ny1 = nx;
y1 = x;

% First, add a delay. I will do this by simply changing the values in nx
% Distance = 100 m; Speed of sound = 340 m/s; Sampling rate = 11025 Hz
ny2 = nx + floor(100*11025/340);
[ny2, y2] = discreteconvolution(ny2, x, 1:length(h), h); % Use the mountain impulse respc
y2 = y2*0.15; % Add attenuation, with A = 0.15

% Add both intermediate signals to a single output signal
ny = (nx(1):ny2(end))'; % y[n] may be non-zero between these values
y = zeros(1, length(ny))';

y(find(ny==ny1(1)):find(ny==ny1(end))) = y(find(ny==ny1(1)):find(ny==ny1(end))) + y1;
y(find(ny==ny2(1)):find(ny==ny2(end))) = y(find(ny==ny2(1)):find(ny==ny2(end))) + y2;

```

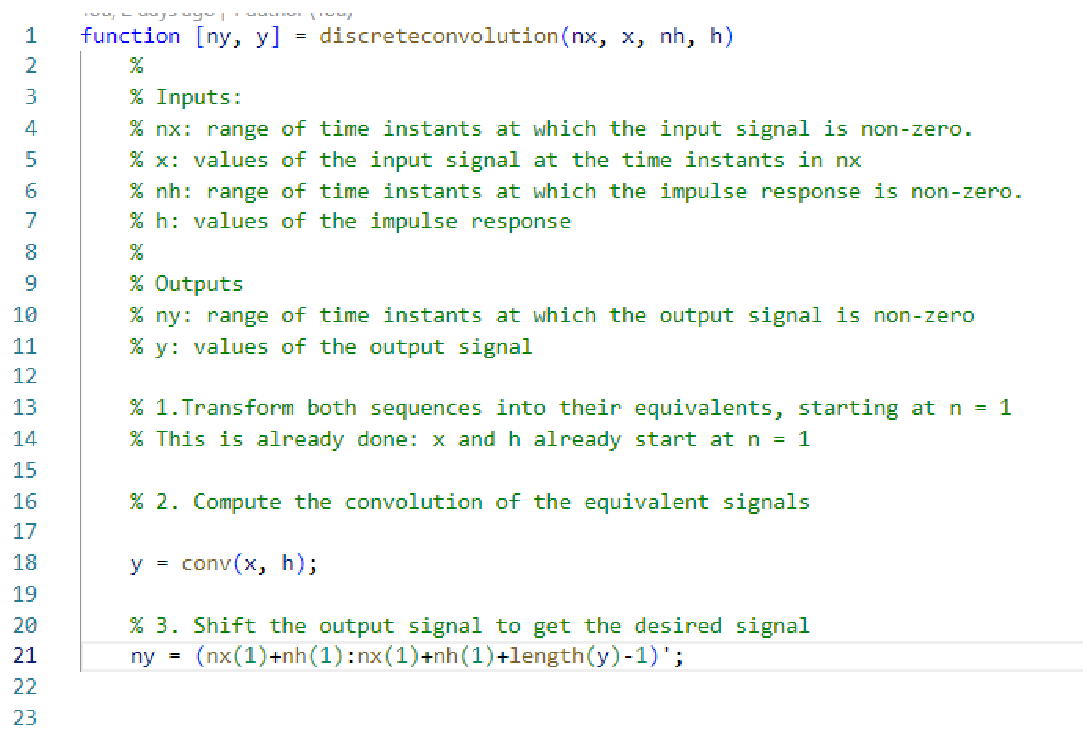
Code:

system2.m



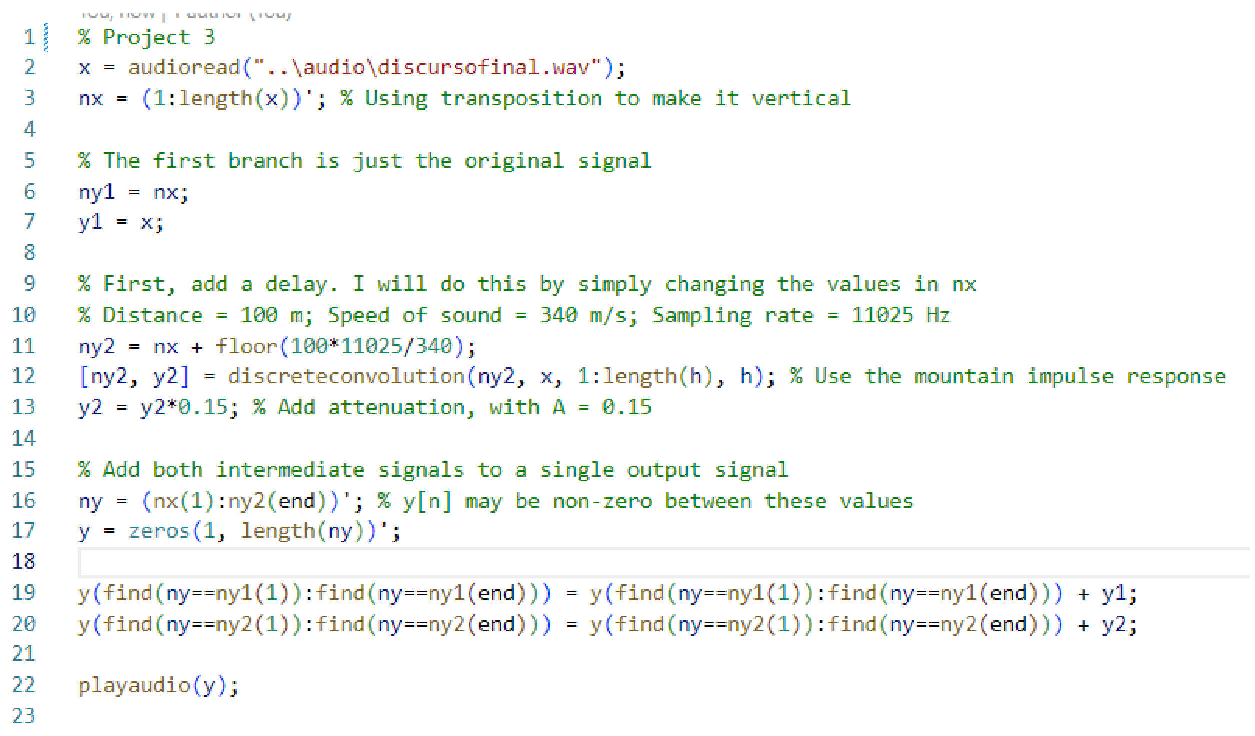
```matlab
function [ny, y] = system2(nx, x)
% y[n] = 0.5x[4 – n] + 0.7x[n – 5] – 0.4 cos(2πx[2 – n])
%
% Inputs:
% nx: range of time instants at which the input signal is non-zero.
% We assume the input signal is zero out of this range.
% x: values of the input signal at the time instants in nx
%
% Outputs:
% ny: range of time instants at which the output signal is non-zero
% y: values of the output signal

% Creating the temporal axis
% I don't really need all the values in between the upper and lower bounds of our support,
% so I'm only going to calculate the bounds of the support of each term
% x[n] is defined between nx(1) and nx(end) <=> nx(1) <= x <= nx(end)
% x[4-n] defined <=> nx(1) <= 4-n <= nx(end) <=> 4-nx(end) <= n <= 4-nx(1)
nyi = [4-nx(end) 4-nx(1)];
% x[n-5] defined <=> nx(1) <= n-5 <= nx(end) <=> nx(1)+5 <= n <= nx(end)+5
nyi = [nyi; nx(1)+5 nx(end)+5];
% x[2-n] defined <=> nx(1) <= 4-n <= nx(end) <=> 2-nx(end) <= n <= 2-nx(1)
nyi = [nyi; 2-nx(end) 2-nx(1)];

% Creating an empty signal
ny = (min(nyi(:,1)):max(nyi(:,2)))'; % y[n] may be non-zero in this range
y = zeros(1, length(ny))';


% 0.5*x[4 – n] term
y1 = 0.5*x(end:-1:1); % x1 is the vector of values of x[4-n] at the time instants in ny1
y(find(ny==nyi(1,1)):find(ny==nyi(1,2))) = y(find(ny==nyi(1,1)):find(ny==nyi(1,2))) + y1;

% 0.7*x[n – 5] term
y2 = 0.7*x;
y(find(ny==nyi(2,1)):find(ny==nyi(2,2))) = y(find(ny==nyi(2,1)):find(ny==nyi(2,2))) + y2;

% -0.4*cos(2*pi*x[2 – n]) term
y3 = -0.4*cos(2*pi*x(end:-1:1));
y(find(ny==nyi(3,1)):find(ny==nyi(3,2))) = y(find(ny==nyi(3,1)):find(ny==nyi(3,2))) + y3;
```

discreteconvolution.m



```
function [ny, y] = discreteconvolution(nx, x, nh, h)
    %
    % Inputs:
    % nx: range of time instants at which the input signal is non-zero.
    % x: values of the input signal at the time instants in nx
    % nh: range of time instants at which the impulse response is non-zero.
    % h: values of the impulse response
    %
    % Outputs
    % ny: range of time instants at which the output signal is non-zero
    % y: values of the output signal

    % 1.Transform both sequences into their equivalents, starting at n = 1
    % This is already done: x and h already start at n = 1

    % 2. Compute the convolution of the equivalent signals

    y = conv(x, h);

    % 3. Shift the output signal to get the desired signal
    ny = (nx(1)+nh(1):nx(1)+nh(1)+length(y)-1)';


```

project3.m



```
% Project 3
x = audioread("..\audio\discursofinal.wav");
nx = (1:length(x))'; % Using transposition to make it vertical

% The first branch is just the original signal
ny1 = nx;
y1 = x;

% First, add a delay. I will do this by simply changing the values in nx
% Distance = 100 m; Speed of sound = 340 m/s; Sampling rate = 11025 Hz
ny2 = nx + floor(100*11025/340);
[ny2, y2] = discreteconvolution(ny2, x, 1:length(h), h); % Use the mountain impulse response
y2 = y2*0.15; % Add attenuation, with A = 0.15

% Add both intermediate signals to a single output signal
ny = (nx(1):ny2(end))'; % y[n] may be non-zero between these values
y = zeros(1, length(ny))';

y(find(ny==ny1(1)):find(ny==ny1(end))) = y(find(ny==ny1(1)):find(ny==ny1(end))) + y1;
y(find(ny==ny2(1)):find(ny==ny2(end))) = y(find(ny==ny2(1)):find(ny==ny2(end))) + y2;

playaudio(y);

```

See all code at https://github.com/alonso-herreros/uni-syscirc-lab2