Unit – Digital Signal Processing – Assignment 1

This assignment involves programming. The recommended programming language is MATLAB and recommended library functions referred to in the problem may be found in its Signal Processing Toolbox. Implementations in other suitable languages, using equivalent library functions (such as the Julia packages Plots and DSP, or the Python packages matplotlib and scipy.signal), are equally acceptable.

The solutions and answers to all parts should be submitted as a PDF file (or alternatively computer printed) and should include all source code written, along with any required outputs produced by the programs. (On MATLAB, the publish function helps to combine both into a single PDF; with Julia or Python, a Jupyter notebook provides similar functionality.)

The completed answers should be submitted either via

https://www.vle.cam.ac.uk/course/view.php?id=174711

or handed in to the Teaching Administration Office (GC04), along with the completed cover sheet provided there, no later than 16:00 on Wednesday 6 November 2019.

Students may be required to sign an undertaking that work submitted will be entirely their own; no collaboration is permitted.
MATLAB commands (similar to)

```matlab
x = [ 0 0 0 -4 0 0 0 0 0 0 2 2 2 ... 
     2 0 -3 -3 -3 0 0 0 0 1 -4 0 4 ... 
     3 -1 2 -3 -1 0 2 -4 -2 1 0 0 3 ... 
     -3 3 -3 3 -3 3 -3 3 -3 0 0 0 0 0 ];

n = 0:length(x)-1;
y = filter([1 1 1 1]/4, [1], x);
plot(n, x, 'bx-', n, y, 'ro-');
```

produced the plot on slide 19 to illustrate the 4-point moving average system. The standard library function `filter(b, a, x)` applies to the finite sequence `x` the discrete system defined by the constant-coefficient difference equation with coefficient vectors `b` and `a` (see slide 25 and “help filter”).

Change this program to generate the corresponding plot for the

(i) exponential averaging system (slide 20)

(ii) accumulator system (slide 21)

(iii) backward difference system (slide 22)

and provide the coefficient vectors `b` and `a` for each of these systems.

(b) (i) Simulate the reconstruction a sampled frequency-limited signal following these steps:

- Generate a one second long Gaussian noise sequence \( r \) with a sampling rate of 300 Hz, where each sample is independent and identically distributed and drawn from a normal distribution, using the `randn` function.

- Taper the noise sequence \( r \) by setting its first and last 15 samples to zero.

- Use the MATLAB function `fir1(50, 45/150)` to design a finite impulse response low-pass filter with a cut-off frequency of 45 Hz. This function will return a vector `b` for use in a digital filter of the type shown on slide 25. What vector `a` is required in addition?

Use the `filtfilt` function in order to apply that filter to the generated noise signal, resulting in the filtered noise signal \( x \). (This function applies the filter twice, once in forward and once in backward direction.)

- Then sample \( x \) at 100 Hz by setting all but every third sample value to zero, resulting in the (equally long) sequence \( y \).
• Implement sinc interpolation with a suitably scaled sinc function to reconstruct the zeroed samples of \( y \), resulting in a reconstructed sequence \( z \).

• Generate another low-pass filter with a cut-off frequency of 50 Hz and apply it with `filtfilt` to \( y \), resulting in interpolated sequence \( u \). Multiply the result by three, to compensate the energy lost during sampling.

• Plot \( x \), \( y \), \( z \) and \( u \), all on top of each other in one figure, and compare \( x \) with \( z \) and \( u \).

(ii) Why should the first filter have a lower cut-off frequency than the second?

(c) (i) Simulate the reconstruction of a sampled band-pass signal, following these steps:

• Generate a 1 s noise sequence \( r \), as in part (b)(i), but this time use a sampling frequency of 3 kHz. Set the first and last 500 samples to zero.

• Apply to that with `filtfilt` a band-pass filter that attenuates frequencies outside the interval 31–44 Hz, which the MATLAB Signal Processing Toolbox function `cheby2(3, 30, [31 44]/1500)` will design for you. [Hint: this function returns two vectors \( b \) and \( a \)]

• Then sample the resulting signal at 30 Hz by setting all but every 100-th sample value to zero, resulting in \( y \).

• Implement sinc interpolation with a suitably scaled and modulated sinc function to reconstruct the zeroed samples of \( y \), resulting in a reconstructed sequence \( z \).

• Generate with `cheby2(3, 20, [30 45]/1500)` another band-pass filter for the interval 30–45 Hz and apply it to \( y \), to reconstruct the original signal as sequence \( u \). (You’ll have to multiply it by 100, to compensate the energy lost during sampling.)

• Plot all the produced sequences and compare the original band-pass signal \( x \) and the two reconstructed versions \( z \) and \( u \) after being sampled at 30 Hz.

(ii) Why does the reconstructed waveform differ much more from the original if you reduce the cut-off frequencies of all band-pass filters by 5 Hz?
Julia DSP package equivalents of mentioned MATLAB library functions (where names differ):

```plaintext
filter ⇒ filt
fir1(50, 45/150) ⇒
digitalfilter(Lowpass(45, fs=300), FIRWindow(Windows.hamming(51)))
cheby2(3, 30, [31 44]/1500) ⇒
digitalfilter(Bandpass(31, 44, fs=3000), Chebyshev2(3, 30))
```

END OF PAPER