Lab 7 – Sampling and Reconstruction

7.1 A signal and its samples

Consider the continuous-time signal $x(t) = \cos(5\pi t) + \sin(10\pi t)$ for analysis. Since continuous-time signals cannot be exactly represented in Matlab, we will use a very fine time-grid to approximate continuous nature of time. Let `t_fine = 0:0.001:2` be the time-grid for representing continuous time-signals (note that in this session `t_fine` is a proxy for continuous-time). Write matlab script for following:

>> Plot this signal as a function of time using the `plot()` command. The time axis should be from 0 to 2s. In this lab we will restrict to the time interval [0, 2] and the time vector `t_fine` will be used in all the tasks below. You should use `plot(t,x)` instead of just `plot(x)` to get appropriate markings on the time axis, else matlab will default to positive integer markings (vector index) which is not informative. Same applies for the `stem()` plots below.

>> Let this signal be sampled with sampling interval Ts = 0.1s and denote the discrete-time signal as $x[n] = x(nT_s)$. In the same figure above, plot the samples $x[n]$ using the `stem()` command in the time interval [0, 2]. Plotting would be easier if you generate the time vector corresponding to the location of the samples: `t_samples = 0:Ts:2`. Use appropriate `t_samples` in the tasks below as well.

7.2 Reconstruction methods

We will make use of the `interp1()` matlab function for reconstruction of continuous-time signal from samples. Read the documentation and examples before using this function.
For this matlab script, repeat part 7.1 above and plot it in the top-left panel of a figure with 2x2 subplots. In each of the following three parts, perform reconstruction as indicated and plot the original samples ($x[n]$) and reconstructed signal in the remaining panels. Use the values of `t_fine`, `t_samples` and Ts from part 7.1.

a) From the samples $x[n]$, perform *zero-order hold* reconstruction of $x(t)$. Use `interp1()` command to get this signal with appropriate selection of the 'method'. Note that the reconstruction signal should be computed over the time-grid `t_fine`.
b) From the samples $x[n]$, perform *linear interpolation* based signal reconstruction of $x(t)$. Use the `interp1()` command.

c) Recall that the ideal reconstruction using sinc function is given by the formula

$$x_r(t) = \sum_{n=-\infty}^{\infty} T_s\, x(nT_s) \frac{\sin\left(\omega_c(t - nT_s)\right)}{\pi(t - nT_s)} \quad \rightarrow (1)$$

Though this is an infinite sum and cannot be exactly implemented in a computer, we will approximately implement it by restricting to the time interval [0, 2] and using only the samples $x[n]$ we have from that interval.

>> Write a matlab function `sinc_recon()` with inputs and outputs as follows:

```
function xr = sinc_recon(n,xn,Ts,t_fine)
% n - the integer locations of the samples x[n]
% xn - the sampled signal x[n] = x(n*Ts)
% Ts - the sampling interval
% t_fine - the time-grid for reconstruction of xr
% xr - the reconstructed signal over the time-grid t_fine
```

From the samples $x[n]$, find the approximate *sinc interpolated* signal as given by the above formula. Always use a cut-off frequency of $\omega_c = \omega_s/2$, where $\omega_s = 2\pi/T_s$.

While manually writing the sinc expression take precaution to avoid divide-by-zero issue (matlab will not flag this but your code will have errors). Alternately, you can use the inbuilt `sinc()` command but with appropriate time scaling to get required cutoff frequency. Read up the documentation for this function before using.

Restrict each of the sinc in the summation to the interval [0, 2].

>> Compare the quality of the three interpolations visually. How does the quality of sinc reconstruction vary within the interval [0, 2]? Give explanation for your observations. >> In your script, for each of the three interpolation methods above, compute the maximum absolute error (MAE) between the original signal and the reconstructed signal in the interval [0.25, 1.75].

>> (Optional): try some of the other interpolation 'method' available in the command `interp1()` and check how the quality of reconstruction and the MAE changes.

7.3 Sampling non-band-limited signal

We know that sampling theorem can be applied only for band-limited signals. All the above tasks had band-limited signals. We now consider a non-band-limited signal and investigate its reconstruction as sampling interval Ts is changed. Write a matlab script for following.

>> Consider the continuous-time triangular pulse signal of height 1, base in the interval [1,1], and zero otherwise. Because this is a time-limited signal, only finite number of terms appear in the reconstruction formula (1) above (though the sinc shape is still infinite in time extent).
>> For a sampling interval of Ts, what is the corresponding `t_samples` vector so that we only consider samples at the base of the triangle (assume there is a sample starting at -1)? Generate the corresponding samples $x[n]$ and the discrete-time indices n. Use these as inputs below.
>> Perform sinc interpolation for this signal using samples generated for the four intervals i) Ts = 0.5s, ii) Ts = 0.2s, iii) Ts = 0.1s and iv) Ts = 0.05s. For reconstruction, use a time-grid of `t_fine = -10:0.001:10`.

>> Create a figure with 2x2 subplots, one panel for each Ts. In each panel plot the samples and the reconstructed signal corresponding to the four sampling intervals. What are your observations as sampling interval is changed?

7.4 Audio signals

Download the 4 audio files given [here.]() Save them in your working folder so that the audio files and code are in same folder. Write a matlab script for the following tasks.

>> Look up the audio file properties and note down its *bit rate*.
>> Look up documentation of the inbuilt matlab function `audioread()`. Use it to load the audio file in to your matlab workspace. What is the *sampling frequency* f_s of these audio signals?
>> From the length of the loaded signal and the *sampling frequency,* compute the duration of each of the audio signals in seconds.
>> From the observed *bit rate* and *sampling frequency*, compute how many bits the ADC must have used while quantizing/storing these signals. How many levels of quantization this ADC can perform?
>> Look up documentation of the inbuilt matlab function `sound()`. This function in combination with appropriate hardware in your computer performs DAC. Use it to listen to the audio files you have loaded in your workspace (ignore the 'nBits' input to this function).
>> For one of the files, listen to the sound using lower sampling frequency than the true f_s

(for example you can use $0.9\ f_s$, $0.8\ f_s$, $0.7\ f_s$, etc. while using `sound()` command). What do you notice?

>> Repeat the above using higher sampling frequency than the true f_s (for example you can use $1.2\ f_s$, $1.\ 4\ f_s$, $1.6\ f_s$, etc. while using `sound()` command). What do you notice?

>> What property of Fourier transform can you use to explain your observations above.

>> Write down your answers as comments at the end of the code.

7.5 Aliasing

Aliasing occurs when the sampling frequency is less than the Nyquist rate required by the sampling theorem. In this matlab script we will look at the effect of aliasing for the signal $x(t)$ = cos ($5\pi t$).

>> What is the Nyquist rate for this $x(t)$?

>> Consider samples of $x(t)$ for the following sampling intervals

i) Ts = 0.1s
ii) Ts = 0.2s
iii) Ts = 0.3s
iv) Ts = 0.4s

For each of these cases perform sinc interpolation from samples over the interval [0, 2].

>> Create a figure with 2x2 subplots, one panel for each Ts. In each panel plot the samples and the reconstructed signal corresponding to the four sampling intervals.

>> What are your observations as sampling interval is changed?