DTU Health Technology, Hearing Systems section
Building 352, DTU, 2800 Lyngby
Phone: 45 25 39 30, Fax: 45 88 05 77 www.hea.healthtech.dtu.dk/

Course 22051: Signals and Linear Systems in Discrete Time

22051 E21

Cloudy...

Bastian Epp: 45 25 39 53 (office), e-mail: bepp@dtu.dk

First steps in Matlab, sampling, data I/O

1 Sampling

This part of the hands-on deals with the implementation and the visualization of a sampled signal. It will illustrate how the mathematical description discussed in the lecture is translated into something the computer can handle. When implementing the code, make sure to keep it general such that you can for example use a different sampling rate without having to change the code at more than one place.

1.1 Plot a sampled sinusoid over time

Consider the following (time-continuous) signal:

$$x(t) = 2\cos(20\pi t + \pi/4) \quad 0 \leq t \leq T$$

- What is the frequency of that signal? What is the phase?

This is the one we want to get into the computer - so we sample it! Generate the digital signal $x(n)$ with a duration of T=4s. (Hint: You can generate a vector in Matlab by the colon operator with the syntax '`<start>:<increment>:<end>`', e.g. `0:0.1:1` will generate a vector from 0 to 1 in steps of 0.1).

- Use a sampling frequency[1] of 200 Hz.
- Plot the sampled sinusoid on a time axis using a line. Add labels on the axes and limit the axes to [0, T].

  `useful commands: plot(xdata,ydata), help plot, doc plot, xlabel('label'), ylabel('label'), xlim(min:max), ylim(min:max)`

- How many samples do you have and how many would you expect?

  `useful command: length, size`

- How can you express the number of samples you get by using the parameters f_s and T?

Great. You managed to generate $x(n)$ as the digitized version of $x(t)$. Now, let's see what happens when we use a lower sampling frequency. If we already have a sampled signal, we can just pick a subset of the samples to get a signal that is sampled at a lower frequency. You can of course also just sample at the new sampling frequency.

- Downsample the signal $x(n)$ to sampling frequencies of 10 Hz, 4 Hz, 2 Hz, 1 Hz and 0.25 Hz by picking the appropriate subset of samples and save them into new vectors x_10, x_4, x_2, x_1 and x_025.

  `useful command: x(<start>:<increment>:<stop>)`

[1] In some books, the expression 'sampling period' is used rather than 'sampling frequency'. The sampling period T_s is simply the inverse of the sampling frequency: $T_s = 1/f_s$

- Plot the signal $x(n)$ with lines (using `plot`) together with each of the downsampled signals using `stem` into the same plot using different colors (this means you will end up with 5 plots).
- Try to show only ticks on the axes at specified positions (e.g. every 1s or every 0.1s)
- Output one of the figures into a file in .png format. When you didn't scale the figure, the font size will be really small and the lines really thin. Follow the instruction on the 'General remarks about the use of Matlab' to scale the figure and do the export again.

  `useful commands: saveas, print`

We can assume that $x(n)$ is a good approximation of the analogue signal. What happens once the sampling rate is being reduced?

2 Data I/O: .wav files

Here signals will be read from .wav-datafiles. These files have the advantage that information about the signal is stored in the header of the file. This information can be read out along with the signal. This part of the hands-on enables you to read (and write) .wav files that can be played and processed further (e.g., your own cell-phone ring tone). Note: file separators differ between operating systems. While Windows-systems use a backslash '\', UNIX systems use a '/'. To keep your code portable, you can use an identifier that both systems will understand: 'filesep'. An example for this is:

```
% DO NOT USE:
% signal_read = audioread('path_to_files\name_of_file.wav');

% BUT USE GENERALIZED VERSION:
signal_read = audioread(['path_to_files' filesep 'name_of_file.wav']);
```

Note that the filename is not written as a single string, but rather as a vector of strings!

2.1 Data loading, plotting, listening and the effect of quantization

Load the signals stored in the following wave files. Plot the signals over time and add the sampling frequency and number of bits in the title of the figure. You might consider writing a function that uses the file name as an input. Make sure the axes are properly labeled and scaled. Provide also (e.g. in two panels next to each other) a magnification of the time signal that seems to be characteristic and/or interesting.

- Pop1.wav
- Sax12.wav
- Hum.wav
- Sweep.wav
- Vuvuzela.wav
- scrambled_eggs_5s.wav

`useful command: wavread (in newer versions: audioread), title`

Once the signals are read into Matlab, you can listen to them (`useful commands: sound, soundsc`). Make sure you that your volume is not turned up too high (especially when using `soundsc`)!

The following files are the same .wav file, but quantized with a different number of bits. Load the files, plot them and listen to them. You can hear how quantization has an impact on the sound quality.

- lecture_01_demo_aliasing_mini-me_short_16bits.wav
- lecture_01_demo_aliasing_mini-me_short_8bits.wav
- lecture_01_demo_aliasing_mini-me_short_4bits.wav
- lecture_01_demo_aliasing_mini-me_short_2bits.wav

2.2 Signal generation and data saving

Generate a so-called tone-complex that consists of a sum of sinusoids that have integer multiple frequencies of each other:

$$s(t) = \sum_{k=1}^{N} a_k \sin(2\pi \cdot k \cdot f_0 \cdot t + \phi_k) \quad k \in \mathbb{Z} \tag{2.1}$$

The sum can be realized in a for-loop. You can use the code written to generate the signal in the first part of the hands-on and sum up the single components to the final signal.

Generate signals (choose values of N between 5 and 100) with equal amplitude $a_k = 1\ \forall k$, fundamental frequency f_0 of 1 kHz at a sampling frequency of 44.1 kHz and duration of 5 s where

- The components are added up "in zero phase", i.e. all have the same phase of zero
- The components all have the same phase of $\frac{\pi}{2}$
- The components have uniformly distributed random phase in the interval $(0, 2\pi)$

  `useful command: rand (for uniform distribution)`

- Is there some limit of N that needs to be considered here?

Save the signals as stereo .wav files (32 bits). Plot the signals and listen to them - is there a difference?

3 Toolbox: Sinusoidal signal with defined amplitude, frequency and phase

Now knowing how to generate sinusoidal signals of all types, it is time to write a proper (and re-usable) function for this. You will be able to (and should) use this function in all upcoming hands-on and assignments, so it is time very well spent to do this properly. Write a function that takes as input arguments:

- amplitude
- frequency in Hertz
- phase in multiples of 2π
- sampling frequency in Hertz
- duration in seconds

and returns

- a time vector
- the time signal

Follow the guidelines on the the use of MATLAB in the guide "General remarks about the use of Matlab" to include a proper help function and an example on how to use the function. It could look something like:

```
function [time_vector signal] = generate_sinusoid(a, f, phi, fs, T_s)
% Function call:
%
% >> [time_vector signal] = generate_sinusoid(a, f, phi, fs, T_s)
%
% INPUT:
% a : amplitude
% f : frequency of sinusoid (in Hz)
% phi_k : phase (in multiples of 2pi)
% fs : sampling frequency (in Hz)
% T_s : duration (in seconds)
%
% OUTPUT:
% time_vector : time vector with sampling points
% signal : the output signal
%
% This function generates a sinusoid at amplitde <a>, frequency <f>, phase <phi>
% sampled at a sampling frequency <fs> and a duration of <T> seconds.
%
% Example:
%
% >> [t sig] = generate_sinusoid(0.1, 1000, 0, 44100, 1)
%
% generates a sinusoid with amplitude 0.1, frequency of 1000 Hz, phase zero
% sampled at a sampling frequency of 44100 Hz and a duration of 1 second.

...here comes the code...
```