ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
School of Computer and Communication Sciences

Handout 1 Modern Digital Communications
Assignment 1 September 9, 2024 (due Sept 17, 2024)

A MATLAB/Python Refresher and AM Signals

In this assignment we do some MATLAB/Python warm-up exercises and we demodulate an AM (Amplitude Modulation) signal.

MATLAB Take advantage of the reference card that you find on the course webpage.[1] Also, take into account the style guidelines discussed in class. Unless otherwise instructed, in this assignment you are allowed to use only standard MATLAB functions (as opposed to toolboxes).

Python NumPy and SciPy provide most of the functions you need to use. We encourage you to browse trough the documentation. The corresponding links are provided on the course webpage. Note that for the second part of this assignment, all the functions you need to implement are grouped under `my_amPlay.py`.

EXERCISE 1. Let $s(t)$ be a baseband signal with support in the interval between 0 and `d` [seconds]. Let `s` consist of its samples taken every `1/fs` seconds, with the first sample taken at $t = 0$.

1. Generate the vector `t` such that `plot(t, s)` produces a plot with the correct time scale on the x-axis.
2. Let `s_f` be the DFT of `s`. Generate the vector `f` such that `plot(f, fftshift(abs(s_f)))` produces a plot with the correct frequency scale on the x-axis.
3. **MATLAB** Implement the function `my_tfplot`. Keep this file somewhere safe, as it will be used again in future assignments.

   Python Implement the function `my_utilMDC.tf_plot`.

Figure 1 shows the output of the function[2] when called like this:

```
[my_]tfplot(m, fs, 'm', 'A sinc signal')
```

(where `m` is the sinc signal shown in the figure).

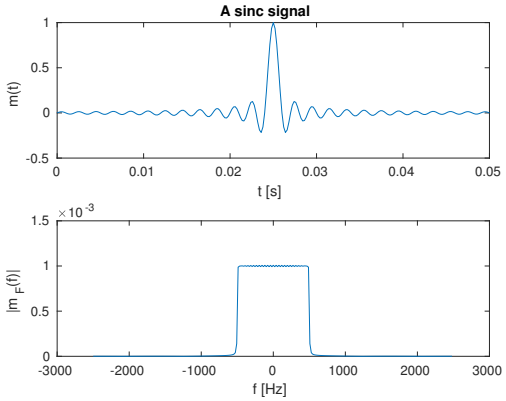

Figure 1: Example output of the function `[my_]tfplot`.

[1] `http://moodle.epfl.ch`

[2] For testing purposes, on the course webpage we provide a function `sincplot` which produces Figure 1. We also provide the solutions for each of the functions you have to implement. In order to test your implementation, you can just replace `[sol_]tfplot` with your own version of `[my_]tfplot`. Note that in our implementation of `[my_]tfplot` we have used a DFT size which is an integer power of 2 (the smallest integer power of 2 which is larger than the required DFT size: in MATLAB, the command `nextpow2` will compute this; in Python, you can easily derive it).

EXERCISE 2.

1. Write a function `s = [my_]ammod(m, K, A, fc, fs)` that modulates the signal `m` using AM, where `fc` is the carrier frequency f_c, `K` and `A` are constants as defined in class, and `fs` is the sampling frequency.

2. In a new function `[my_]test_am()`, create the message signal

$$m(t) = \frac{1}{2}\cos(2\pi f_{\text{info}} t),$$

   and its corresponding AM signal, where $f_{\text{info}} = 10\,\text{Hz}$, $f_c = 300\,\text{Hz}$, $A = K = 1$, and $f_s = 4000\,\text{Hz}$. Let the duration of the signal be $d = 1\,\text{s}$. Use `[my_]tfplot` to display the message signal as well as the modulated signal. Do the plots correspond to your expectations?

3. Write a function `md = [my_]amdemod(s, fc, fs)` that demodulates the AM signal `s` and returns the message signal `md`, normalized to have values between -1 and 1. As before, `fc` is the carrier frequency and `fs` is the sampling frequency. We recommend that you use the averaging method discussed in class.

   You might find useful the MATLAB function `movmean`. Alternatively, you can convolve your signal with a box function of suitable width. You can use a moving average over 10 carrier periods. Test your function by calling it from within `[my_]test_am()`. Plot the result using `[my_]tfplot` and verify that the message signal has been correctly reconstructed.

4. Now modify the parameters of the message signal $m(t)$ to produce a more audible signal, and use `sound` to play both the original and the reconstructed message signal to verify that they are the same. Note that we do not necessarily use the default sample rate of the command `sound` (you should check the help page).

5. Once you feel confident that your code works and that you understand what is going on, load the file `am_data.mat` which contains the samples of an AM signal, along with the values of the carrier frequency `fc` and the sampling frequency `fs`.

   Write a function `my_play_am()` that loads the signal from the file, demodulates it using `my_amdemod`, downsamples[3] the result by a factor 10 and plays it using `sound` (again, for this command you should pay attention to the sampling rate).

As for the previous exercise, in order to help you with the implementation, we provide on the course website the solutions of the functions you have to write.

MATLAB We provide you with a script named `function_mapper`, that you can use to choose between your implementation and the solutions of `my_ammod.m` and `my_amdemod.m`. The script `function_mapper` is called from within `sol_test_am.m` and `sol_play_am.m`. If you wish to use it within your own scripts or functions, you need to include the command `function_mapper;` in your code before calling `hw1_ammod.m` or `hw1_amdemod.m`.

Python You can implement the moving average function using for example `numpy.convolve` in an appropriate way. If using `playsound` does not work, write the demodulated data to a .wav file using `scipy.io.wavfile.write` and then play it with an audio player.

[3] cf. the MATLAB function `downsample`