1 Introduction

This set of exercises is meant to review some basic features of Matlab which will be useful in the future homework. There is no grade for this Matlab exercise.

If you are unfamiliar with Matlab, there are several sources of on-line help available. The most important ones are:

- Typing `help topic` displays help about that topic if it exists.
- Typing `lookfor keyword` searches and returns a list of topics and commands that contain the keyword you specify. The keyword need not be a Matlab command and case does not matter.
- The `helpwin` command open a new window that can be used to view the help text for Matlab commands in an interactive Help window that uses mouse for navigation.
- A wide range of Matlab help and reference information can be viewed using the Help Desk facility via a Web browser. Typing `helpdesk` to start this browser-based help.
- In addition, you could consult a short Matlab guided tour called Matlab Primer available on the Web (e.g. via the Google search engine).

2 Matlab Programming

2.1 M-files and M-functions

M-files are like shell-scripts. Type a bunch of Matlab commands, save the file as `filename.m` and each time you type `filename` from the command prompt, that bunch of commands is executed. M-files take variables form the current workspace, so they can access and modify all the global variables.

M-functions are just like m-files, save that they begin with a line like

```
function [a, b, c, d] = foobar(x, y, z)
```

which means that the function `foobar` takes three input arguments and returns four. Assignments like `a = 2` are the way to return values to the calling environment. Also, m-function do not have access to the current workspace and do not modify global variables (unless instructed to do so).

2.2 Vectors and Matrices

When writing Matlab code, one has to think in matrix notation as much as possible. While at first this goes against the old C code practice, yet one should try to write as few for loops as possible. The reason is that Matlab m-files are interpreted, which implies a lot of overhead for constructs like loops; built-in array functions like matrix and vector multiplication, on the other hand, are very fast.
Exercise 1 – Circulant matrices

Write an m-file which takes a vector $x$ as input and builds a circulant matrix $A$ with $x$ as the first row (always without using loops – hint: look up the `toeplitz` command). While you’re at it, build also the DFT matrix $D$ of the same size, and verify that

$$D \ast A \ast \text{conj}(D)$$

is diagonal. Remember that the $N \times N$ DFT matrix has entries

$$D_{nm} = e^{-j2\pi nm/N}.$$ 

3 Matlab and DSP

3.1 Basic Signals

In the physical world, signals are continuous time, continuous amplitude. Within Matlab we could say we almost have continuous amplitude given that all computations take place in double precision, but there’s no escape from discrete time. This means that we always deal with finite vectors, indexed by integers, and sometimes it’s easy to lose contact with the physical meaning of these quantities (admitting they had one to start with).

There are two cases in which we will have to juggle with continuous time and frequency and discrete indices: in defining sinusoidal signals and in looking at spectra. The idea is that there always is an underlying sampling frequency $f_s$ for our vectors, which will almost never pop out explicitly.

The easiest continuous time sinusoidal signal is

$$x(t) = \cos(2\pi ft)$$

where $f$ is expressed in good old familiar Hertz and $t$ in seconds. In Matlab, to get the first $N$ samples of this continuous time signal at the sampling frequency $f_s$, we can use

$${x = \cos(2\pi f_s [0: N-1]/f_s);}$$

If we plot the signal using `plot(x)`, the x-axis would just show the indices of the vector $x$ (starting from 1). To see the real time $t$ in seconds, use

`plot([0: N-1]/f_s, x)`

Given our $N$ sample vector $x$, and performing a DFT on it, we will have another $N$ sample vector $y = \text{fft}(x)$. Similarly with the time domain, if we plot its magnitude `plot(abs(y))`, the x-axis would just show the DFT indices. To see the real frequencies as you would on a spectrum analyzer, use

`plot(f_s*([0: N-1]-N/2)/N, abs(fftshift(y)));`
Exercise 2 – Indexing

Generate a 16384-sample vector of a 256 Hz sinusoid, sampled at 8192 Hz. Play the data using `sound` – it should be a middle C. Plot the spectrum of the signal.

3.2 Fourier Transforms

In Matlab, which always operates on finite vectors of discrete samples, when we’re talking Fourier Transform we always mean discrete Fourier Transform (DFT); the DFT changes a finite $N$-point sequence into another finite $N$-point sequence (just a change of base in $\mathbb{R}^N$).

Computing the DFT in practice is a no-brainer: just use the `fft` command. More often than not, however, we will want to have $N$ equal to a power of two for efficiency reasons. What people generally do then, is to zero-pad their sequence prior to transformation, which means adding zeros to it until it becomes $N$ point long. It is important to understand the consequences of this harmless operation. Basically, zero-padding in the time domain corresponds to Lagrange interpolation in the frequency domain. That is to say, we are not adding information and the things we could not see with the DFT of the original sequence, we will not see in the DFT of the zero-padded sequence.

Exercise 3 – DFT and zero-padding.

Create 32-point vector $x$ which can be though of the first 32 samples of

$$x(t) = \cos(2\pi 1000t) + \cos(2\pi 1100t)$$

sampled at $f_s = 8000$ Hz. Plot the DFT of the vector plus different amounts of trailing zeros and verify that, no matter how much zero-padding, you cannot resolve the two sinusoids. How many true samples of $x(t)$ do you approximately need before you can see the two spectral peaks?

3.3 Linear and Circular Convolutions

Linear convolution can be performed in two ways: `conv` convolves two given vectors implementing the standard sum-of-products; `filter`, on the other hand, implements a general difference equation. In the case of FIR filtering, the filter coefficients are the same thing as the filter’s impulse response, so that the methods are equivalent. Look up both commands in the on-line help for details.

Circular convolution implemented via FFT can speed up things when we have a short signal which is to be convolved with a much longer one using techniques such as overlap-add. Some care has to be taken, though; given two sequences $x_1$ and $x_2$ of length $L$ and $P$ respectively, their circular convolution will yield samples equal to their $(L + P - 1)$-point linear convolution only if the period of $N$ of the circular convolution (and therefore the number of points for the DFT) satisfies

$$N \geq L + P - 1;$$

$N$ is usually chosen as a power of two while $x_1$ and $x_2$ are zero padded to $N$ points. See [Oppenheim and Schafer, pages 548–560] for details.

Exercise 4 – Convolutions

Write an m-function `cconv(a, b)` which takes two vectors $a$ and $b$ of any length and computes their convolution using the `fft` command; the function must give exactly the same result as `filter(a, 1, b)`.
4 Wavelet ToolBoxes

There is a rich set of wavelet toolboxes, which partly help wavelets became such a widely used tool in signal processing and data analysis. The Wavelet Toolbox from Mathworks is the easiest and most convenient one since it comes with Matlab. In addition, it provides an excellent Graphical User Interface (GUI) to explore the various aspects and applications of wavelets. The WaveLab package by Donoho and co-workers is the most powerful one, since it contains the richest set of Matlab routines for wavelets and related time-frequency transforms, together with plenty of workout scripts for studying wavelets.

Exercise 5 – A Wavelet Demo

Start the Wavelet Demo by typing wavedemo. Select GUI mode and then Wavelet 2-D. Step through the demo to see the famous decomposition process that you just saw in the lecture! To find out what the Matlab commands that have been used, rerun the same demo in command-line mode (by selecting Command line mode and then Wavelet 2-D from the Wavedemo menu).

Now try to reproduce this experiment yourselves using GUI tools. Start the wavelet GUI by typing wavemenu from the command line. Select Wavelet 2-D from the main menu to load up the corresponding window. Next you need to load an image so that it can be processed. The easiest way is to select from the File menu the Demo Analysis option and then select a particular wavelet system together with an image, for example At level 2, with Haar ---> Woman. You can change the used wavelet and number of levels later using the Wavelet and Level buttons. Play around with different tools, settings and observe the results.