Massachusetts Institute of Technology
Department of Electrical Engineering & Computer Science

6.341: Discrete-Time Signal Processing

Fall 2005

Project Assignment

Issued: Sept. 27, 2005 Project I due: Nov. 1, 2005 Project II due: Dec. 6, 2005

This handout contains all parts of both projects (Project I and Project II) to be assigned in 6.341 this semester. The writeup for Project I is due on November 1, 2005. The writeup for Project II is due on December 6, 2005. Both projects are required to be done, turned in by all students independent of which homework option you chose, and will be counted in the final grade.

You are required to turn in written reports for both Projects I and II. Project I will have two separate writeups, one for Part A and one for Part B. Project II will have one writeup. Each writeup should have a summary of how you went about it (anywhere from one to five pages, not to exceed five pages). In addition to the five page maximum, you can include any appropriate Matlab plots. Do NOT turn in any Matlab code. The total number of pages including description and plots for Project I Part A should not exceed five pages and for Project I part B should not exceed 10 pages. The total maximum number of pages for Project II is 10 pages.

No formal solutions to the project will be handed out. We may select one or two of the reports handed in and (with the permission of the student) will distribute those to the entire class. Each writeup (two for Project I, one for Project II) will receive a grade which reflects our assessment of your level of understanding of the various issues involved and the effort that you put into it. For the actual grading, we will be using the "EGRMU" grading scheme which is an acronym for E(xtraordinary), G(ood), R(easonable, kind of), M(arginal), U(nsatisfactory). We are likely to use some mixed grades like E/G etc. In generating a numerical grade at the end of the semester, we will use the following scale: E=100, E/G=90, G=80, G/R=70, R=60, R/M=50, M=30, M/U=15, U=0.

COMPLETION OF THE PROJECTS AND WRITEUPS
IS A REQUIREMENT TO PASS THE COURSE.

Note: MATLAB help sessions will be held by the TAs to help you get started on the project. This means that the TAs will be available in a cluster and will be open to questions. These help sessions are intended for MATLAB- and project-related questions and are not to be used as general office hours.
Also note that, to encourage you to start early on the project, the last problem in problem set 5 asks about your progress on the project.

Playing Sound in MATLAB

Throughout these projects it will be necessary to play audio files which you will have manipulated in MATLAB. While the function `sound` works well for doing this on Mac (OS X) and PC platforms, using `sound` on Athena machines is problematic and should be avoided. The staff has instead written several functions which should be used to play sound in MATLAB on Athena Sun (UNIX) machines in lieu of `sound`. (Athena Linux-based machines are not supported within the context of this project.) These functions, `usound441` and `usound110`, are available on the course website as `.m` files and will play back sequences at sampling rates of 44100 Hz and 11025 Hz, respectively.

To install, simply put `usound441.m` and `usound110.m` in a directory which is in your MATLAB path, or in your working directory. For information on how to use the functions type `help usound441` or `help usound110`. Note that you may need to type `add infoagents` at your Athena prompt before starting MATLAB for these functions to work properly.

Overview

One area in which discrete-time signal processing has enjoyed widespread use is in the field of speech processing. In this set of projects we consider various issues relating to applying digital filtering to speech. Specifically, in the first part of Project I, we consider the effect of all-pass filtering on speech. Your current background in 6.341 should be sufficient to allow you to begin on this part immediately. In the second part of Project I we consider IIR and FIR filter design in the context of enhancing speech corrupted by additive noise. While this relies somewhat on material in Lectures 7-8, some aspects can be started earlier. In Project II we continue the second part of Project I, with a constraint imposed on the allowable multiplier rate (number of multiplies per input sample) and the effect of this on different filter designs. Although Project II isn't due until December 6, it does not rely on lectures beyond L10 and we concourage you to start on it as soon as possible.

Historically, DSP courses have put considerable emphasis on design techniques for both IIR and FIR filters, as discussed in Chapter 7 of the course text. A variety of filter design algorithms are now implemented in common software packages such as MATLAB. This makes it unnecessary for most practitioners to learn the details of the design algorithms; however, ceding the design function to a software package makes it *more* important to understand the properties of different types of

optimal filters, the meanings of the design parameters, and some of the trade-offs between filter classes.

In the context of what we are covering this semester on IIR filter design, we suggest that you read Section 7.0 including Example 7.1. Also, read the examples in Section 7.1.3, but focus on what Butterworth and Chebyshev filters are rather than on the details of the bilinear transformation. For the discussion of FIR filter design, the suggested reading is Sections 7.2, 7.3, and 7.4 through 7.4.1. We will not be going into the details of the issues raised in Section 7.4.1, but we would recommend at least reading that section to get a sense of what some of those issues are.

Project I: (Due November 1, 2005)

Project I. A.

A common "folk theorem" states that the ear is insensitive to phase, i.e. that for audio, phase distortion is inaudible. If that is correct, then processing audio with an all-pass filter should not result in perceived distortion. This part of the project tests this conjecture. The all-pass filter that we consider is of the form

$$H(z) = \left[\prod_{k=1}^{3} \frac{(z^{-1} - e_k^*)(z^{-1} - e_k)}{(1 - e_k z^{-1})(1 - e_k^* z^{-1})}\right]^N = \left[\frac{\sum_{k=0}^{6} b_k z^{-k}}{\sum_{k=0}^{6} a_k z^{-k}}\right]^N .$$

For this part of the project, the file `projIA.mat` will need to be first loaded into MATLAB. After downloading this file from the course website, copy it into your MATLAB working directory and type `load projIA`. The b_k and a_k coefficients above are stored in the variables `b` and `a` respectively; `b(`$k+1$`)`$= b_k$ and `a(`$k+1$`)`$= a_k$.

(a) Implement the all-pass filter for $N = 1$ and show plots of the impulse response (the first 100 samples), the magnitude of the frequency response, and the group delay.

(b) Plot the pole-zero diagram. You may find the function `roots` helpful in doing so.

(b) Process the speech file, stored as the variable `speech` (sampled at rate `fs` = 11025 Hz), using the command `filter` to run the filter implemented in (a). Listen to the result and comment specifically on whether there is audible distortion.

(c) Implement the filter for $N = 50$ and again show plots of the impulse response (the first 5000 samples), the magnitude of the frequency response, and the group delay.

(d) Repeat (b). In your writeup, comment specifically on how you would subjectively describe the distortion and on what aspect of the filter is it's primary cause.

Project I. B.

For this part of the project, the file `projIB.mat` will need to be first loaded into MATLAB. After downloading this file from the course website, copy it into your MATLAB working directory and type `load projIB`.

In this part of the project, it may be helpful to use MATLAB's order estimation functions (e.g. `buttord`, `cheb1ord`, ...). A caveat in the use of these functions is that MATLAB's definition of 'ripple' differs between the IIR and FIR filter design functions. The following can be used to sort through these differences in convention:

1. In the design of IIR filters, MATLAB takes specifications for ripple magnitude and stopband attenuation as input parameters. A specification of ripple magnitude δ_{IIR} yields an IIR filter design with a maximum passband gain of unity (or 0 dB) and a minimum passband gain of $10^{-\delta_{\text{IIR}}/20}$ (or $-\delta_{\text{IIR}}$ dB). A specification of stopband attenuation ξ_{IIR} yields an IIR filter design with a maximum stopband gain of $10^{-\xi_{\text{IIR}}/20}$ (or $-\xi_{\text{IIR}}$ dB).

2. In the design of FIR filters, MATLAB also takes a specification of ripple magnitude as an input parameter. A specification of ripple magnitude δ_{FIR} yields a FIR filter design with a maximum passband gain of $(1+\delta_{\text{FIR}})$ (or $20\log_{10}(1+\delta_{\text{FIR}})$ dB), and a minimum passband gain of $(1-\delta_{\text{FIR}})$ (or $20\log_{10}(1-\delta_{\text{FIR}})$ dB). A specification of stopband gain ξ_{FIR} yields an FIR filter design with a maximum stopband gain of ξ_{FIR} (or $20\log_{10}(\xi_{\text{FIR}})$ dB).

Clearly, to compare FIR and IIR filters designed with MATLAB, we must find a correspondence between our desired specifications and the input parameters to the filter design functions. Let us thus define $G_{pb_{max}}$ (dB), $G_{pb_{min}}$ (dB), and $G_{sb_{max}}$ (dB) to refer to the desired maximum passband gain, minimum passband gain, and maximum stopband gain, respectively. In order to have free control over each of these parameters, we also introuce k_{FIR} and k_{IIR}, linear scaling terms which are used to normalize our FIR and IIR designs.

Before beginning Project I.B. do (a) and (b) below and include your solutions as part of the project writeup.

(a) Find δ_{FIR}, ξ_{FIR}, and k_{FIR} in terms of $G_{pb_{max}}$, $G_{pb_{min}}$, and $G_{sb_{max}}$. These parameters will be used for FIR designs.

(b) Find δ_{IIR}, ξ_{IIR}, and k_{IIR} in terms of $G_{pb_{max}}$, $G_{pb_{min}}$, and $G_{sb_{max}}$. These parameters will be used for IIR designs.

Project I.B. is concerned with designing a low-pass filter for the removal of high-pass noise from a speech signal. The noisy signal is stored in the variable `noisy` and was sampled at 44100 (`fs`) Hz. It consists of a summation of a speech signal which was band-limited at 4 kHz using a low-pass filter with a very narrow transition band, and a noise signal which was filtered at 4 kHz using a high-pass filter with a very narrow transition band. To filter the noise we must design a discrete-time filter with the following parameters:

1. Passband edge: 2500 Hz.

2. Stopband edge: 4000 Hz.

3. Maximum gain in the passband $G_{pb_{max}}$: 40 dB.

4. Minimum gain in the passband $G_{pb_{min}}$: 37 dB.

5. Maximum gain in the stopband $G_{sb_{max}}$: -55 dB.

Because it may take a few iterations to get each filter right, we suggest you write a .m MATLAB script for each filter.

Note that in this part of the project, you may end up with very high order filters since the specifications are rather severe. You can try to implement the IIR filters according to the specs using MATLAB's built in tools (such as the `fdatool` or the command line tools `butter, cheby1, cheby2`, and `ellip`—type "`help signal`" for more information). If you do so, you will notice that the resulting systems might not be stable. There are several reasons for this, the most important of which being coefficient quantization—even with floating point precision. This issue arises because the specifications are very tight and some of the filter types have all of their poles concentrated near $z = 1$.

Fortunately, there is a workaround. What you need to do is to group the poles and zeros for the desired filter in pairs (conjugate or not) to create smaller stable second order filters of the form $N(z)/D(z)$ where $N(z)$ and $D(z)$ are at most second order polynomials in z. A cascade of such filters will produce the desired system, and MATLAB will be able to analyze it. The drawback is that you will have to implement your own methods to generate the plots and simulate the system.

Design a DT filter of each of the following types based on the specifications given above:

Butterworth, Chebyshev Type I, Chebyshev Type II, Elliptic, Parks-McClellan, Kaiser.

For each of the designs, provide the following in your Part I project write-up:

(a) The order of the filter.

(b) The number of multiplication operations per input sample required to implement the filter. Be sure to explain the structure you assume.

(c) Plot the magnitude response (in dB) from $\omega = 0$ to $\omega = \pi$ using `freqz`. Plot a detail of the magnitude response, focusing on the passband ripple (linear scale). Plot the group delay (in samples) using `grpdelay`. (Use `subplot` to fit the three plots on the same page for each filter.)

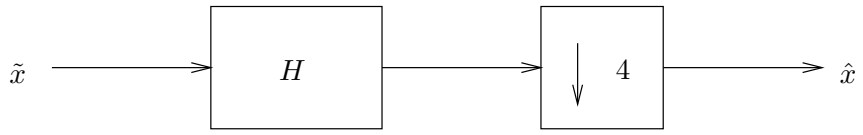
(d) Plot the pole-zero diagram.

(e) Plot the impulse response using `filter` and `stem` for 100 samples. (Use `subplot` to fit the pole-zero diagram and the impulse response on the same page.)

Filter `noisy` using your de-noising filter. Listen to the filtered and original files. How do they compare?

Project II: (Due December 6, 2005)

In this project, you will design filters for a "hardware" implementation of the de-noising system in Project I B. Since the de-noising filter does a good job of keeping the resulting signal band-limited to 4 kHz, it is prudent to add a compressor-by-4 (which for no aliasing requires its input to be band-limited to 5512.5 Hz) as a final stage in the implementation of our system. The sampling rate at the output of the system is thus 11025 Hz, and the sampling rate at the system's input is still 44100 Hz. The overall de-noising system, which operates on a noisy signal $\tilde{x}$ to yield a cleaned signal $\hat{x}$, is depicted in its nominal form below.



The hardware on which we would like to implement the de-noiser is indeed very limited; a maximum of **17** multiplications can be performed per input sample. (The 18th multiplier is used elsewhere to implement a digital volume control.) In an attempt to lessen the constraints on our design problem, we are told that we must adhere to all of the parameters outlined in Part I.B. **except** passband edge, which should be kept as close to the stopband edge as possible, while still meeting the multiplication constraint. Our new design criteria become:

1. Maximum number of multiplications per input sample: 17
2. Passband edge: as close to 4000 Hz as possible.
3. Stopband edge: 4000 Hz.
4. Maximum gain in the passband $G_{pb_{max}}$: 40 dB.
5. Minimum gain in the passband $G_{pb_{min}}$: 37 dB.
6. Maximum gain in the stopband $G_{sb_{max}}$: -55 dB.

Again design a DT filter of each of the following types based upon these new specifications:

Butterworth, Chebyshev Type I, Chebyshev Type II, Elliptic, Parks-McClellan, Kaiser.

Note that for this section, writing a `.m` script will almost certainly prove useful. For each of the designs, provide the following in your Project II project write-up:

(a) The order of the filter.

(b) The passband edge parameter chosen.

(b) The number of required multiplications per input sample, both with and without the use of polyphase. Were there filter implementations which did not meet the required specification for *any* arbitrarily low choice of passband edge?

(c) Plot the magnitude response (in dB) from $\omega = 0$ to $\omega = \pi$ using `freqz`.

Now implement your de-noising system using each filter you designed. For designs where a polyphase implementation is assumed for purposes of multiplication counting, feel free to do the actual filtering in MATLAB on the high-rate side of the compressor, remembering that a polyphase implementation implies only a specific filter *form* and will therefore not change the overall behavior of the system. To implement a compressor-by-4 on a sequence `x` in MATLAB, use

```
x = x(1:4:length(x));.
```

On the course website you will find a list of unique `.wav` files which contain speech utterances in the presence of noise. Download the file corresponding to your Athena username and run it through the de-noising system you implemented, using each filter you designed. (You can read `.wav` files into MATLAB using `wavread`.) In your writeup, specifically comment on the following: How does the quality compare for each filter design? What aspects of each filter do you think contribute to this? What does your utterance say? Feel free to experiment with other users' utterances as well.