GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING

ECE 4271 SPRING 2016
MATLAB FINAL PROJECT

Adaptive Equalization

Assigned: Wednesday, April 13, 2016
Due Dates:

Report: Tuesday, April 26, 2016 @ *beginning* of lecture (3:05 PM)
Matlab Code and Report: Wednesday, April 27, 2016 by 11:59 PM in t-square

- This project is to be done *individually.* Collaboration on projects is *not* OK. Each student must work on the projects independently. Each student must develop his or her own analyses and computer code in its entirety. Any project-specific help needed should be sought from the TA and Dr. Marenco. Students are not to discuss the theory or approaches to coding the theory with one another, nor are they to assist in debugging each other's work. ***The Georgia Tech honor code and its policies apply.***

  You may ask Dr. Marenco or the TA questions regarding theory and implementation of the project, including asking them at the beginning or end of class or during office hours, when others can benefit as well.

- Data required for this project are available for download from t-square in the Final tab located in the Resources section. You will also find reference materials and code there, if any. ***Whether you begin working right away or not, be sure you download the data and make sure you can load it into MATLAB as soon as possible to avoid last minute difficulties.***

- Reports and code will be graded primarily on completeness in addressing the assignment and quality of results. Reports must be typed, not handwritten; should be well-organized; and must clearly explain your design and the decisions and analysis that went into it. Code must be provided in the form specified below, and should be commented well-enough to be clearly understandable.

- Questions or clarifications should be directed to Dr. Marenco[1] or to Gukyeong Kwon (gukyeong.kwon@gatech.edu) . Errata, revisions and hints (if any) will be made available via e-mail, t-square site, or during class.

[1] e-mail: <alvaro.marenco@gtri.gatech.edu>

1. INTRODUCTION

In digital communications one of the great problems is inter-symbol interference (ISI). ISI can be caused by channel distortion. For example, a common type of distortion is channel dispersion as a result of the nonlinear phase characteristics of the channel. In this case the received sequence y[n] is

$$y[n] = \sum_{k=-\infty}^{n} x[k]h[n,k] + w[n]$$

where x[k] is the transmitted sequence, h[n, k] is the time-varying impulse response of the channel, and w[n] is additive noise. (For this example we can assume that w[n] is zero–mean, white Gaussian noise.) The result of this distortion is that the signal is received with errors. Since the channel distortion is unknown and possibly time-varying, the equalizer is typically an adaptive filter.

For this task, a message sequence is encoded, then modulated and sent through a dispersive, noisy channel. Your objective is to decode with minimum error a series of files. Please refer to Section 5 for more details.

2. TASK

Write a MATLAB program that accepts as input a `column vector x (e.g. M x 1, where M is the number of samples)` containing a digital data sequence representing a quadrature phase shift keying (QPSK) signal corrupted by noise and channel dispersion (NOTE: you should have now familiarity with QPSK symbols as we worked in Project #2 – your signal x is complex). Your processing architecture must use a version of the LMS adaptive filtering algorithm, but may vary in the details of the algorithm (*e.g.*, normalized or not, *etc.*) and the other functions and steps that you use. You must write your own LMS filtering code; you may not use code obtained from any other source.

3. REQUIREMENTS

You must submit a report of your methods and findings that must include:

1. An overview description of how your processing algorithm works. This must include a block diagram of the major processing steps with a description. The accompanying text should describe the rationale for each step.

2. The text should explain the analyses and design decisions that led to each of your parameter choices, such as filter order, filter initiation strategy, adaptive constant μ, training strategy, *etc.*

3. The text should include "significant" snippets of your MATLAB code (don't include every line) that implement each step.

4. Include "before" and "after" constellation plots of the data for the lowest SNR case on which you are "reasonably successful". (See Section 5 for a discussion of this criterion.) Do this separately for the noise-only, time-invariant dispersion, and time-varying dispersion data groups (three pairs of "before" and "after" plots.) If you are not reasonably successful on any of the data sequences, give the plots for the best case you have in each group.

5. Include a plot of your final filter weights. Plot the real and imaginary parts separately.

6. At the end of the report, include a complete listing of the MATLAB program used to process the signals. Code must be sufficiently modular and well-commented to be understood readily. Listings must also be included for any functions used, except for those which are built into either MATLAB or its Signal Processing Toolbox, or were provided in class.

A hard copy of your report should be handed in at the beginning of class on the day it is due. In addition, you must submit your MATLAB code AND the written report in t-square by the report-code deadline (April 27, 2016 – 11:59 PM). See below for further instructions on naming the zip file and how your code must be organized. Your program will be tested by applying it to test signals generated by Dr. Marenco (*may not be* the same signals provided for development of your algorithm.)

Start working early. If you encounter problems, please ask questions to clarify anything that is vague or incomplete.

4. REFERENCES

I will look for some reference papers and, if I find any suitable ones, will post them in t-square in the *Final* tab located in the *Resources* section. However, the class materials should be adequate to complete the task.

5. DATA

Sample data sequences are available in the Winzip file `AdaptiveFilteringFiles.zip` available from t-square in the *Final* tab located in

the *Resources* section. When unzipped, this will produce a `README.txt` file and a `gettysburg.mat` file containing several complex data sequences, as well as some additional files. `Read the README.txt` file for information on the various files and signals. The signals are the received binary data stream after QPSK encoding, passage through a dispersive channel, and addition of noise. The noise is complex valued, zero mean, and with independent but equal variance in the real and imaginary channels. Each signal encodes a non-trivial amount of text (specifically, Abraham Lincoln's "Gettysburg Address"). Thus, a correctly decoded sequence, when displayed, should reproduce the original written text (except in lower case, as discussed below).

The text characters have been encoded into binary in such a way that only 5 bits per character are needed (by now you should be familiar with these directions from Project #2). This was done by limiting the characters to the following:

- space, comma, period, apostrophe, line feed, return, and the 26 *lower case* characters 'a' through 'z'

Because of the lower case limitation, your output text will be entirely in lower case characters. The specific encoding is as shown in the following table:

Character	ASCII Decimal Value	Binary Code in This Project
space	32	00000
,	44	00001
.	46	00010
'	39	00011
line feed	10	00100
carriage return	13	00101
'a' through 'z'	97-122	00110-11111

The original text is pre-pended with the training sequence and two spaces, and padded at the end with a few spaces. After this encoding, original text data containing L characters will be represented by a binary sequence of $M = 5L$ bits.

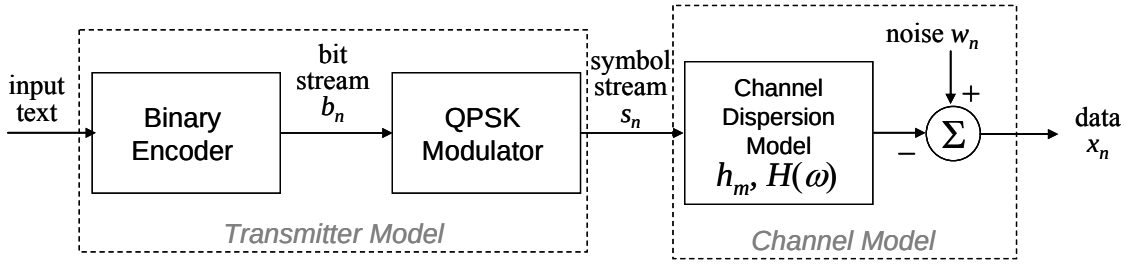
You may find MATLAB commands such as `char`, `dec2bin`, `bin2dec`, `num2str`, `str2num`, `double`, and `reshape` to be useful. (I am *not* saying you *have* to use all of these, or even any of them; but some may be helpful.)

QPSK takes successive pairs of bits and represents them as one of four "symbols", namely $\pm 1 \pm j$ according to

$$s_n = (-1)^{b_{2n}} + j(-1)^{b_{2n+1}}$$

where s_n is the n^{th} symbol and b_{2n} and b_{2n+1} are successive even- and odd-numbered bits in the bit stream. L characters thus require $\lceil 5L/2 \rceil$ QPSK symbols.

The data stream x_n that you are given to decode will have passed through the following system:



The channel dispersion model will be a time-varying nonrecursive (FIR) filter, so that the data x_n can be represented as

$$x_n = \sum_{m=0}^{M-1} h_m[n]\, x_{m-n} + w_n$$

Note that the dependence of the filter coefficients h_m on n indicates that the filter impulse response is time-varying. Consequently, the filter **must continue** to adapt throughout the signal reconstruction process. This case is particularly important when the signal under test has time-varying characteristics.

To recover the text, you must pass the signal x_n (this is the signal I will pass to your function – QPSK symbols) through an LMS adaptive filter, a QPSK demodulator, and a binary-to-text decoder. Your code should return a single string variable that contains the entire decoded signal. My test routine will display this variable on the screen, and I will judge the quality of your results according to how accurate the reconstruction is (*i.e.*, how many typographical errors it contains) as a function of SNR.

It might be a good idea to use the signal `s` in the data file, which is the QPSK symbol stream with no noise and no dispersion, to debug your QPSK decode and binary-to-text converter before you start dealing with the extra complications of the noise, dispersion, and adaptive filtering. Again, a case similar to this situation is what you worked in Project #2.

Even if you don't start work right away, be sure to download your data file(s) and make sure you can load and work with it as soon as possible!

6. SIGNAL PROCESSING ISSUES

Following are some considerations and questions you must deal with in designing your decoding algorithm.

1. *LMS Filter Order*. You must select an order for your LMS adaptive filter. The order of the channel dispersion model will be "small". You should experiment with your filter order to see what size is necessary for good results.

2. *LMS Filter Type and Adaptation Rate*: You must choose to implement a basic LMS filter, normalized LMS, or some other variant, and then to pick an appropriate value of μ for your filter. Be sure to document and explain your choices.

3. *Training:* You must develop a training strategy. The data has a limited number of symbols (think about the number of symbols, not characters) at the beginning for which the ideal (noise-free, undistorted) symbol sequence is known; however, you must assume the ideal symbol sequence is unknown after this point because the corresponding text is unknown. Also, a strategy that works for time-invariant dispersion may not be adequate for time-varying dispersion.

4. *Filter Delays*: You must pay careful attention to the handling of delays in your system to make sure that the filtered output samples form the LMS filter are compared to the right samples from the training sequence.

5. *Initial Filter Weights*: You must use some rule to initialize the filter weights, such as using random weights, using all-zero weights, are letting the initial impulse response be $h[k] = \delta[k]$ ("no-op" filter). You may want to try more than one choice to see if it affects performance.

6. *Counting Errors*: "Reasonably successful" means the text is at least 90% correct. Let `tx` be the text generated from `x` (the data before filtering) and `ty` be the text generated from `y` (the data after filtering). An easy way to count the number of errors is `length(find(tx ~= ty))`. However, you should skip over the characters corresponding to the training sequence at the beginning of the text, since errors are likely in the early stages of filter adaptation.

7. MATLAB CODE SUBMISSION AND EVALUATION

Your MATLAB code must be submitted in t-square in a single zip file before the project deadline. ***No other submission method is acceptable!*** The following conditions apply to this zip file and your code:

1. The file must contain all of the MATLAB code, including all functions and subfunctions, data files, *etc.* required to run your algorithm on a test signal. Place all of these files in a single flat (no subdirectories inside of it) directory with the name `FirstInitialLastName.zip`. For example, my file would be named `AMarenco.zip`. Do not include any functions that are built into MATLAB or its Signal Processing Toolbox.
2. Your code should have your name in the comments in the first few lines of the file.
3. Your top level function *must* have as its first line

   ```
   function message = lms_decode(x)
   ```

   Furthermore, it must work when all of the files in your zip file have been unzipped and placed in the MATLAB `work` directory. If you develop code on another system, *e.g.* Octave or similar, port it to a Windows or Unix MATLAB implementation in the ECE or CoC computing labs and make sure it runs correctly in that environment.
4. Your program should produce an output that is a single string variable. For instance, if the original data was derived from the original text "The quick brown fox jumped over the lazy dog", your output should be a single string variable message such that, if I give the command above, what I will see on my command window is the following (notice I did not put a semicolon at the end of the first line):

   ```
   >> message = lms_decode(x)
   the quick brown fox jumped over the lazy dog
   >>
   ```

   (Notice also that the first word "The" turned into "the" in the output; remember that only lower-case letters are encoded in the signal, so do not be surprised by this. It is not an error.)

   The actual signals you are provided will include a training sequence at the beginning of the data, so your output will also include the training sequence.

Also, do not print the output variable message within your code; I will print it after your function returns, as shown above.

5. I will test your program by simply executing the command

```
lms_decode(x)
```

using various test signals as the input variable `x`. ***Be absolutely certain that your program will run correctly when I unzip your files into the MATLAB `work` directory and type this command into MATLAB.***

My test will consist of running your algorithm with several corrupted sequences representing various text samples with decreasing signal-to-noise ratios. The purpose of this test is to see at what noise level your algorithm "breaks" (starts generating typographical errors). All algorithms will break at some noise level, so I'm not saying in advance what breakage level I consider really good or really bad.

8. GRADING

Your grade on this project will be allocated as follows:

- Code Performance: 70%
    - Performance on a series of decreasing-SNR signals: 50%
    - No minor edits by me required to make it run: 10% (Anything I can't fix in 5 minutes or less will be returned to you for re-work, with penalties to be determined.)
    - Output format correct: 10%
- Report: 30%
    - Algorithm and parameter description and rationale, including code snippets: 10%
    - General report quality (complete but succinct, appropriate figures included (remember the filter weight and constellation plots), writing, *etc.*): 10%
    - Code listing and readability: 10%