UNIVERSITY OF UTAH
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

ECE1020 **COMPUTING ASSIGNMENT 6**
N. E. COTTER **MATLAB® LOGICAL OPERATORS: DECODING**

READING
Matlab® Student Version: learning Matlab 6, Ch 4-25 to 4-27
Mastering Matlab® 7, Ch 10

TOPICS
Relational and Logical Operators

OVERVIEW
When our rover receives a transmitted word from the base station, it must decide which codeword the base station sent. In advanced decoding schemes, the receiver evaluates each incoming bit and assigns it a probability for whether it was a one or zero. The set of codewords determines which sequences of zeros and ones are possible. The problem of decoding reduces to the problem of finding which codeword is closest to the received word, given that each received bit is assigned a probability *between* zero and one rather than a purely zero or one value.

One way of performing decoding is to measure the distance between the received word and every word in the code book. This approach is computationally expensive, but it is easy to understand. In contrast, if the code is carefully constructed, Viterbi decoding, (a version of dynamic programming), finds the closest codeword far more efficiently. The Viterbi approach is also used in speech recognition to find the closest sequence of words to a given utterance.

For the sake of simplicity, we will consider comparing the distance between the received word and every word in the code book. We will also simplify the decoding process by making so-called hard decisions and assuming that each received bit is either a zero or one. When we receive a bit, we decide immediately whether it was a zero or one. Noise will usually flip one or more bits in the message, and our task is to determine which codeword is closest to sequence of zeros and ones we have received. Thus we wish to compute the distance between binary numbers representing codewords and the received word.

Although there are several ways to measure this distance, a particularly simple way is to count how many bits differ in the two binary words. This is called the Hamming distance. When we find the codeword closest to the received word, we have decoded the incoming signal. We hope to get the correct answer!

PROCEDURE

In this assignment, you will use logical operators in the process of calculating the Hamming distance between a received binary word and binary codewords.

Script File for calculations

- Using a text editor program on your PC, create a script file called **hamming_dist.m** containing matlab commands to perform the calculations in this assignment.

- Do **not** use semicolons at the ends of commands in your script files.

+5 pts Create an empty array called received_data.

+5 pts Display a message indicating that the next answer will be a one if received_data is an empty array.

+5 pts Use a function to verify that received_data is indeed empty.

+5 pts Set received_data equal to itself concatenated with an array of 6 random numbers uniformly distributed between 0 and 1. Use the rand function to generate the random numbers. Note that this is rand rather than randn.

+5 pts Use the greater-than operator to quantize all the bits in received_data to zero or one values in one command. Values less than or equal to 0.5 will become zeros, and values greater than 0.5 will become ones. Place the resulting six bits in an array called quant_data.

+5 pts Using similar steps, create a 4x6 array called rand_data containing random numbers uniformly distributed between zero and one. Then quantize the bits to zero and one values, but this time set values less than or equal to 0.5 to ones and values greater than 0.5 to zeros. Place the result in an array called code_words. The rows of this array will be our supposed codewords.

+5 pts Use repmat to replicate the quant_data array as four identical rows of a variable called quant_array. We do this so we have four copies of our received data that we can compare with the four codewords.

+5 pts Use the xor operator to create an array called code_match containing ones where quant_array and code_words have different bit values. Note that the xor operator gives a value of 1 if bits are different and 0 if they are the same.

+5 pts Write Matlab code to count the number of ones in each row of code_match. Place the result in a horizontal array called hamming_nums. These are the hamming distances between the received word and the codewords. If we were complete the decoding process, we would pick the code word with the fewest ones.

+5 pts Use the == operator and the ~ operator to create an array called code_match2 containing the same result as code_match. This is an alternate way of determining which bits differ between quant_data and code_words.

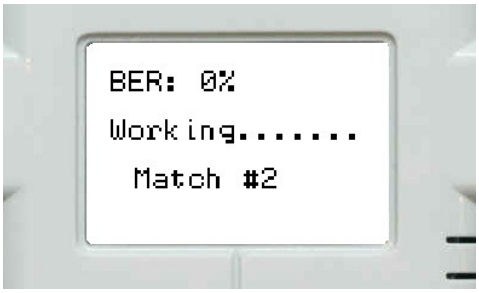
+5 pts Use the "any" function to determine if there is a perfect match between quant_data and any of the code_words. Place the results in a horizontal array called perfect_match. A one represents a perfect match. A zero represents an imperfect match. This decoding method would only work when there happened to be a perfect match.

+5 pts Suppose there is a glitch, caused by an alpha ray, that sets a bit of quant_data to Nan (not a number). Write a comment in your script file describing what effect this has on the decoding process. Do we get an error (crashing our rover program)? Does the nan act like a certain bit value? Does it make a difference whether we use xor or ==?

Rover Implemenation

The decoding of incoming bit streams can be a challenging undertaking but by using the hamming distance as a decoding measure we have simplified the task of matching messages with codewords. The nxt program *randcommand.rxe* can be used generate random commands from the codewords stored on the NXT. By pressing the orange button on the NXT after execution of the program a 20% bit error rate in introduced. This simulates bits being flipped by corruption of the received signal.

+5 pts Add code to connect to the NXT and start to program *randcommand.rxe*



The program saves a random command in the mailbox #10 of the rover to be retrieved for use in Matlab.

Use the function

[message, inbox_number] = readmailbox(nxt, mailbox_number ,0,1)

to retrieve commands to be decoded in Matlab. The retrieved message will be a binary string stored vertically in the variable *message*.

+5 pts Add a pause between starting the program and reading the commands so that there is time to accumulate five commands in the que of mailbox #10 (~20 secs pause) and create an index, i =1.

+5 pts On one line, read a command using *readmailbox()*,store the command in a cell array called *commands*, use the index as a reference to the message number and then increment the index by one. (hint: use comma separation for each command)

+5 pts Use this last line four more time to provide five commands to the *commands* cell array. Transposed the cell array *commands*.

Stop the program on the NXT and Paste following the codewords into you script file

code_words = [0 0 0 0 0 0; 0 0 0 1 1 1; 0 0 1 0 1 1; 0 1 0 1 0 1; 1 0 1 0 1 0; 1 1 0 1 0 0; 1 1 1 0 0 0 ; 1 1 1 1 1 1]

+5 pts Create an array called it *match_index,* to hold an index that indicates a row number in *code_words* which matches the received command. Create another array called *hamming_dist* to store the values of the hamming distance for each command to its closest codeword. Initialize both arrays to zero with the same length of the *commands* cell array

Create an index called *n* and set it equal to 1

+5 pts Convert *commands(n)* from a binary string to a decimal using bin2dec() and store it in a variable called recv_mess. Create a matrix that has 8 rows each filled with *recv_mess* called *recv_mess_mat*

+5 pts Compute the hamming distances between each code_word and recv_mess_mat Use the *min*() function to search the computed distances to find the smallest distance and store it in *hamming_dist(n)*. The index for *code_words* will be stored in *match_index(n)*

Add a line to increment n by 1

Assignment wrap-up

Run Script File
Run your script file by typing the name of the file without the .m
>> hamming_dist
If you make any changes in your **hamming_dist.m** file, be sure to run the following Matlab command to insure that Matlab reads your file again the next time you run it:
>> clear all

+5 pts Finish processing all of the commands by using the **Evaluate Selection** command in the editor window to run the block of code that starts at computing recv_mess and ends with the increment of *n.* Do this four time to finish processing n=2,3,4,5

Verify that the command indexes match the commands received

End of Diary
>> diary off % Close the diary file. Look for the diary in e.g., c:\matlab\work directory.

E-mail your script file (hamming_dist.m) and your diary file to your TA, (as two separate e-mails). In the Subject line of your e-mail, be sure to put Your Name, "ECE1020 Comp6," and the file name, (e.g. hamming_dist.m). Also, print out the files and hand them in to the TA or to the ECE1020 locker.