UNIVERSITY OF UTAH
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

ECE1020 — **COMPUTING ASSIGNMENT 4**
N. E. COTTER — **MATLAB® ARRAYS: CODEWORDS**

READING

Matlab® Student Version: learning Matlab 6, Ch 4.
Mastering Matlab® 7, Ch 6

TOPICS

Array manipulations

OVERVIEW

We have seen that noise can alter the information we send over a communication link. If we transmit in binary—ones and zeros—some bits will be flipped. In this assignment, you will explore error-correcting codes that allow us to correct many of these errors. Such codes are utilized in communication links of all kinds, from space probes to cell phones.

With a error correcting code, we spread out the information we transmit so that if any one bit gets lost in the noise, we still succeed in communicating our message. To understand how this works, consider an analogous situation of trying to communicate an important message by sending emails that may or may not be confiscated en route to their destination. To make sure our message gets through, we might send five emails, each containing every fifth letter of the message. If one message fails to get through, the recipient would probably be able to fill in the missing letters.

An even better scheme would be to spread the message out over several emails *and* add redundancy. The redundancy might increase the cost, however, so there might be a limit to how much redundancy we could afford. For our rover, this translates into sending a different set of bits than the set of bits we directly want to communicate. We also send more bits than we would need to communicate in a noise free system. This raises the following question: Is there a way to correct *all* the errors in communication?

In the 1940's Claude Shannon considered this problem of reliable communication from a purely mathematical standpoint and proved the remarkable Channel Capacity Theorem [1]. In everyday language, this theorem says that it is indeed possible to correct all the errors in transmission if the rate at which we transmit bits is less than the following quantity:

$$C = B \log_2 \frac{P+N}{N}$$

where C ≡ channel capacity in bits/second
B ≡ bandwidth of transmitter
P ≡ average transmitter power in Watts
N ≡ average noise power in Watts

What is perhaps even more remarkable than the theorem is the method of proof. Shannon showed that if one considers all possible ways of encoding bits into code words the average code will achieve the channel capacity. This suggests that a randomly generated code is likely to perform fairly well in correcting all errors, but finding the optimal code among all possible codes is a problem that remains unsolved! Furthermore, the proof of the theorem assumes that arbitrarily long code words may encode arbitrarily long blocks of data bits.

In communication theory, the search continues for codes of finite length that achieve near-optimal performance. We, however, will pursue the more modest goal of creating random codewords.

PROCEDURE

You will manipulate arrays to create error-correcting codes in this assignment.

+5 pts Script File for calculations
Using a text editor program on your PC, create a script file called **errcode.m** containing Matlab commands to perform the calculations in this assignment.

+5 pts Do **not** use semicolons at the ends of commands in your script file.

In the following steps, you will create random 6-bit long code words of ones and zeros.

+5 pts Using concatenation of arrays and the "zeros" and "ones" functions, create an array called zeros_and_ones containing six entries: three zeros followed by three ones.

+5 pts Using only a one-line statement, create a two-dimensional array called perm_six with four rows and six columns with each row equal to the digits from 1 to 6 in random order. Use the "randperm" function once each time you generate a row. (Type >>help randperm for information about randperm.)

+5 pts Use the following command line in your **errcode.m** file, and add a comment that explains exactly what this command line does:

```
rand_codewords = zeros_and_ones(perm_six)
```

+5 pts Use the "size" function in a one-line statement to set variables called number_of_codewords and length_of_codewords equal to the number of rows of rand_codewords and the number of columns of rand_codewords, respectively.

You have now created a codebook for data transmission. The four codewords correspond to the four possible two-bit-long data sequences: 00, 01, 10, and 11. To use this codebook in practice you would chop a data stream into two-bit chunks. The stream 011000101011 would become 01, 10, 00, 10, 10, 11. for example. Then you would transmit the 6-bit code word for 01, then the 6-bit code word for 10, and so on.

+5 pts To simulate the codebook lookup process, create an array called binary_data containing two entries equal to either zero or one. (You may choose any of the four possible two-bit-long data sequences: 00, 01, 10, or 11.) Create a one-line statement in your **errcode.m** file that translates the entries in binary_data into a number from 1 to 4 and then uses this number to extract the corresponding row of rand_codewords, placing the result in an array called transmit_word. Write the statement so that it would work for any of the four possible values for binary_data.

+5 pts The previous steps created four six-bit code words at random. For the sake of practice extracting from arrays, create a 2x4 array called center_code containing the 3rd through 6th bits of the 2nd and 3rd rows of rand_codewords. Use only one statement, and use the colon operator somewhere in that statement.

+5 pts (We now have code words that we will use in later assignments. For the remainder of this procedure, we practice array manipulations under the guise of creating more code words.) Use the randn function to create an 6x6 matrix called rand_matrix containing (normally distributed) random numbers. Then use the logical function abs(?) > 0.5 as part of a Matlab® statement that creates a 6x6 array of ones and zeros called new_codewords. You must determine what should replace the ? in this statement. Also, you must add a comment in your file explaining exactly what this entire command line in your file does and indicate to data type of the resulting matrix.

+5 pts For practice, use the fliplr function to flip the new_codewords matrix in the left-right direction. Store the results in new_codewords2.

+5 pts Use the diag function to create an eight bit codeword called single_codeword equal to the diagonal of new_codewords2. Note that diag applied to a matrix extracts the diagonal, whereas diag applied to a vector creates a square matrix with the elements of the vector down the diagonal. These are very different things!

+5 pts Using the abs function and subtraction, find a way to create a matrix called inverted_codewords that is equal to new_codewords with bits inverted: zeros changed to ones, and ones changed to zeros.

ROVER IMPLEMENTATION:

Now that we have a set of code words to work from we can to implement them on the rover. In some communication schemes the code words are sent as part of the message sent to the device to ensure security. If the code words always remained the same control of the rover could become compromised over time. In the code you have written the code words generated will be different each time you run the script. It makes sense to provide the code words to the rover as part of our scheme.

Check out the NXT rover and use the "Getting started with Matlab and MindStorms" document on the class website to setup up the Bluetooth connection to be used by Matlab.

+5 pts Create an instance of an NXT rover by typing

>>nxt = nxtInterface('COMx') *where x is the com port of the nxt link on your computer.*

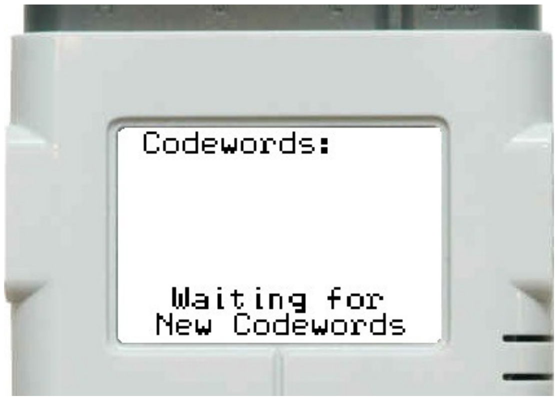
The rover should respond with

>>Opening Serial port (COMx) ... This may take a few seconds

+5 pts Now that a connection has been made we will command the rover to start a program call *codewords.rxe* on the NXT to accept the generated code words. Type the following

>>startProgram(nxt, 'codewords.rxe')

The display should show an empty set of code words and indicate it is ready for the new code words (see figure).



To send code words to the rover we will use a custom command

writemailbox(nxt, '*codeword*' , *mailbox_number*)

This command sends a string message to the rover using one of the ten mailboxes (0-9) on the NXT. To use this function the code words must be converted to binary strings (for example '01110100'). We can use the function *dec2bin*() to convert from rand_codeword's double array to a character array of 1s and 0s. The array of characters must then be transposed using (') to provide the desired string.

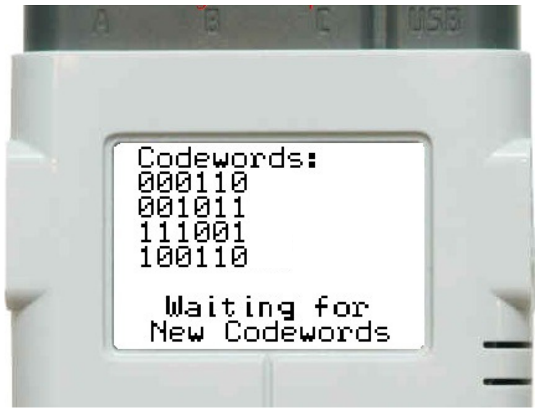
+5 pts Convert each of your code words to a binary string by typing

code_xy = dec2bin(rand_codewords(row_number, :)) ' *where xy = (00,01,10,11)*

Note, that you choose a row to convert using row_number and the ':' provides all the columns as input. The Transpose operator makes the result a string of characters.

+5 pts Write code to send each of the code words to the rover using *writemailbox()*. Send code_00, code_01, code_10, and code_11 to mailboxes 1, 2, 3, 4 respectively. There will be no response from the rover indicating success. To update the code words on the rover send '101010' to mailbox '0'

>> writemailbox(nxt, '101010', 0)



```
Codewords:
000110
001011
111001
100110

 Waiting for
New Codewords
```

The rover should exclaim "hooray" and display the codeword one at a time and then prepare for a new set to be sent (see picture).

Now to clean-up and disconnect form the rover type:

```
>>stopProgram(nxt)
>>delete(nxt)
```

+5 pts Compile the commands used to program the code words onto the rover as part of your script, **errcode.m**. You will need to add a *pause*(10) command to be able to see the program work on the rover otherwise the program codewords.rxe will start retrieve the code words and then abort and delete quickly.

Assignment Wrap-up

+5 pts Diary Command

When you have finished typing the script file for the above equations, delete the current diary file and start a new diary.

```
>>diary on  % turns on diary
```

Change the working directory to the c: drive where you stored your script file.

This an alternative to the addpath command if everything you write is stored in one directory.

```
>> cd c:\
```

+5 pts Run Script File

Run your script file by typing the name of the file without the .m

```
>> errcode
```

If you make any changes in your **errcode.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 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 (errcode.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 Comp4," and the file name, (e.g. errcode.m). Also, print out the files and hand them in to the TA or to the ECE1020 locker.

REFERENCES

[1] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication.* Urbana, Il: University of Illinois Press, 1978, p. 100. ISBN 0-252-72548.

[2] Bryan Stenquist, *Getting started with Matlab and MindStorms.* Online at http://www.ece.utah.edu/~ece1020/MatlabMindstorms_setup.pdf