HW7: Error Detection

CSE1010 Fall 2012

Jeffrey A. Meunier
University of Connecticut

1. Introduction

In this project you will simulate the transmission of data over a noisy communications channel, and the subsequent detection of transmission errors. The steps include:

1. Get input to transmit (just a single character).
2. Convert the character to a vector of binary digits (also known as bits).
3. Add a parity bit to the bit vector, to be used for error detection.
4. Transmit the bits (not really; just flip some of the bits randomly with some low probability to simulate transmission over a noisy communication channel).
5. Determine if a the bits contain an error.
6. Decode the bits into the original character – or whatever character the bits represent now.

In this project you will not do any *error correction*. That will happen in the project subsequent to this one. In this project you will be concerned with merely the detection of the presence of a transmission error

2. Value

This program is worth a maximum of 20 points. See the grading rubric at the end for a breakdown of the values of different parts of this project.

3. Due Date

This project is due by 11:59PM on Sunday, Nov 25, 2012. Note that this is **the end of Thanksgiving break**. I can nearly guarantee that you will not be in the mood to work on this project on the Sunday you return from break. Neither will you be in the mood to work on this project over break, in spite of the best ambitions you have before you leave for break. **Pretend that this project is due on Friday, Nov 16, just before you leave for break**, even though it's not actually due until the 25th. I'm not giving you an extension on this just because you ate too much turkey. Got it? Good.

4. Objectives

In this assignment you will get more experience with vectors and functions, and learn about parity as an error detection mechanism.

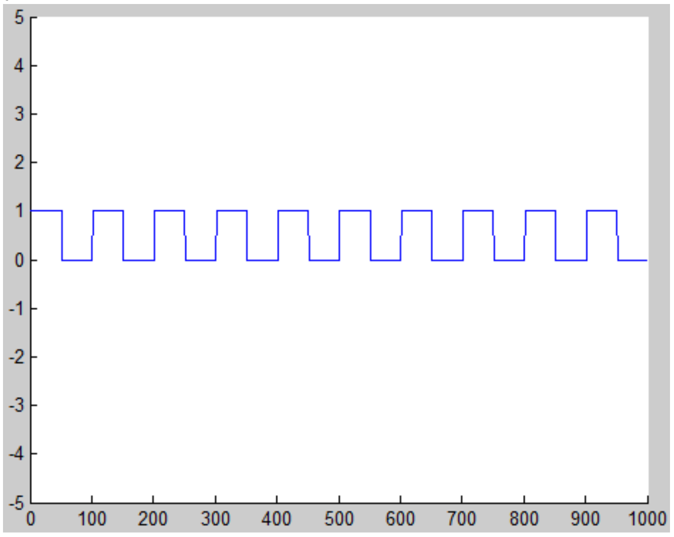
Keywords: vectors, matrices, parity

5. Background

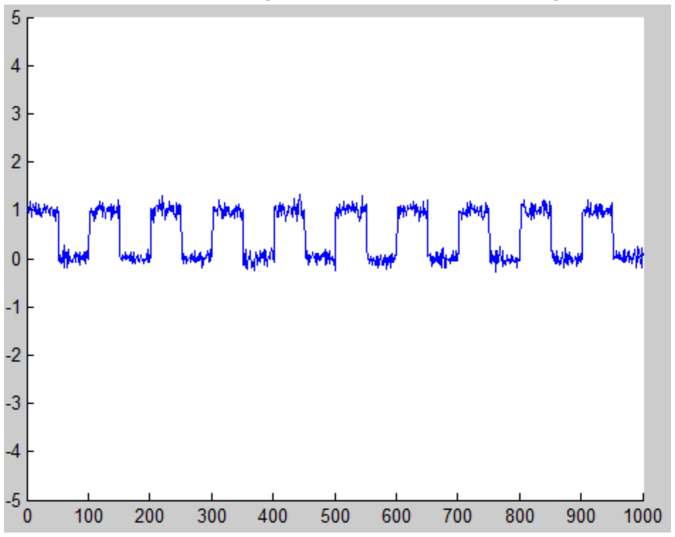
Here is some information you need for this project.

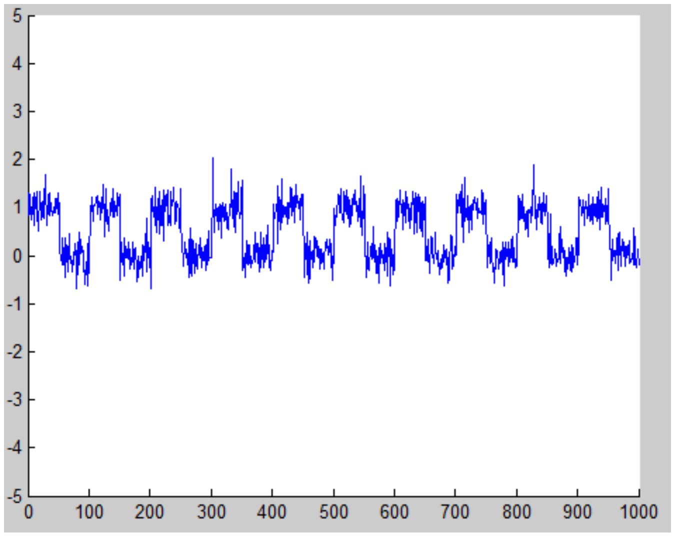
5.1. Signals and noise

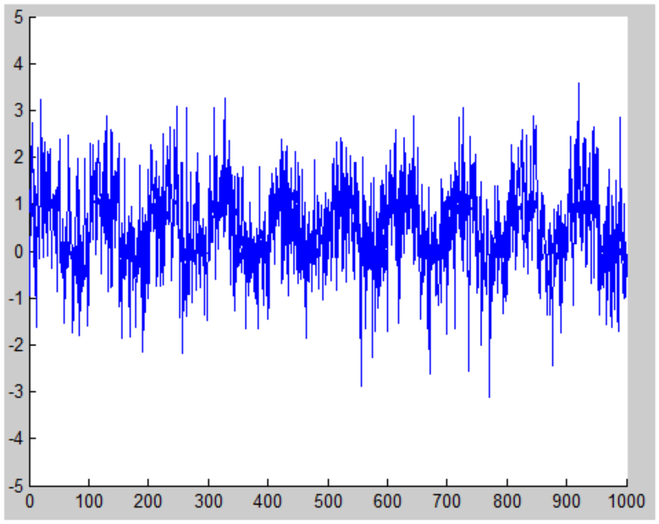
Here is a noise-free digital signal consisting of alternating 1s and 0s. The signal level is the y axis, and time is the x axis.



The next three images show the same signal with varying degrees of noise added:







In the last image it is becoming difficult to determine where the 0s are and where the 1s are, and there is almost no indication that the signal was originally a square wave of pure 0s and 1s.

Furthermore, the noise added to this signal is well behaved noise: the amplitude is random but uniform, and there is no time shifting. A real signal suffers from even more noise problems than this.

5.2. Binary number storage and transmission

All information in a computer is stored in binary. The computer's memory chips store only 0s and 1s in the form of voltage charges that are on or off. In most programming languages like Matlab, when a number is displayed it is first retrieved from memory in binary, then converted from binary into base 10, then displayed as the base 10 number. So when we talk about converting a number into binary, we just mean that we would like to see what the number looks like the way it is stored in memory. Confused? Good!

One way to have Matlab show you the binary format of any decimal number is by calling the **dec2bin** function, but you must be aware that the result is returned as a character string.

```
>> dec2bin(32)
ans =
100000
>> class(ans)
ans =
char
```

When a computer transmits information to another computer, it sends the information in binary form, just a sequence of 0s and 1s. That's because at the most basic level, communications media (wireless network, wireless network, fiber-optic network, etc.) can handle only on/off signals.

5.3. Parity

Read this article from Wikipedia about what a parity bit is (up to the section *Error detection*), but before you do please continue reading here about the XOR operator. In the article the authors use the caret symbol ^ to mean *exclusive or*, which is also written *xor*. This table describes the *xor* operation:

input x	^	input y	=	output z
0		0		0
0		1		1
1		0		1
1		1		0

Notice that the output (or result) z is true (or 1) only if one of the inputs is true. If both inputs are false or both are true, then the output is false. Looking at it another way, z makes the number of 1s in each row even.

5.4. ASCII codes

Read this web page for the definition of ASCII.

6. Assignment

Your program will follow the basic outline given in the introduction.

Write the functions shown below. Be sure to place internal "help" comments in each of your functions.

6.1. Function getChar

The first thing the program must do is get a character from the user.

This function has no parameters. It asks the user to enter a character, then uses the **input** statement to get a string from the user and store it in a variable. Use the help facility in Matlab to determine how to force the input to be a string. Using Matlab's **input** statement it is not possible to restrict how many characters the user enters, so that's why you need to write this function. This function lets the user enter any number of characters, but it must return only the first character that the user entered. Note that a string is just a vector. Test that your function works correctly:

```
>> getChar
Enter a character: a
ans =
a
>> getChar
Enter a character: abcd
ans =
a
>> getChar
Enter a character: 100
ans =
1
```

Be careful: the last string that I entered, 100, looks like a number but it's just a string of the three characters '1', '0', and '0'.

6.2. Function char2bin

Before transmitting the character over the simulated transmission stream, it must be converted to a vector of bits.

This function accepts a character and returns a vector of 8 logical values (bits) that are the binary equivalent of the ASCII value of the character. You will need to use these functions:

- **dec2bin**: takes a number or character and returns its binary representation as a string
- **sprintf**: use the format string '%08s' to force a string to be 8 characters long, and padded with leading 0s
- **logical**: this function will convert a vector of double 0s and 1s into logical 1s and 0s

Note that you will need to convert ASCII codes of the characters '0' and '1' into the values of those digits, 0 and 1.

Digit character	**ASCII numeric value**
'0'	48
'1'	49

Do not use an **if** statement do convert from a character to its ASCII value. You can do it using pure math. Consider this: If you have the character '0' (which in Matlab is the same as the number 48), how would you convert that 48 into a 0? In other words, how do you get from 48 to 0? How do you get from 49 to 1? Or for that matter, how would you get from 50 to 2? The relationship is the same in each case.

Test your function.

```
>> char2bin('a')
ans =
  0  1  1  0  0  0  0  1
```

That is the binary representation of the ASCII value of the character 'a', which is the same as the number 97.

```
>> char2bin('0')
ans =
  0  0  1  1  0  0  0  0
```

That is the binary representation of the ASCII value of the character '0', which is the same as the number 48.

```
>> char2bin('Z')
ans =
  0  1  0  1  1  0  1  0
```

Note that the vector returned by this function *must* contain logical values, not double values.

```
>> class(ans)
ans =
logical
```

Use the **logical** function to convert numbers to logical values.

You can find ASCII tables and tables of binary numbers all over the internet if you want to check your function with additional characters.

6.3. Function bin2char

This function does the opposite of the **char2bin** function, but thankfully it's a bit simpler.

1. Convert the vector of bits to a vector ASCII numbers (0 becomes 48, 1 becomes 49).
2. Convert the vector of ASCII numbers to a character string (use the **char** function).
3. Use the **bin2dec** function to convert the string into a single ASCII number.
4. Convert the ASCII number into a character.

Here's an example that shows that **char2bin** and bin2char are complimentary functions:

```
>> char2bin('X')
ans =
  0  1  0  1  1  0  0  0
>> bin2char(ans)
ans =
X
```

6.4. Function parityOf

Before transmitting the vector of bits, a parity bit must be created. If you skipped the background information on parity, now would be a really good time to go back and read it. Go ahead. I'll wait.

This function has two parameters: a vector of bits, and a desired parity. The vector of bits will be a vector of logical values, just like what is returned from the **char2bin** function. The desired parity will be the number 0 (for even parity) or 1 (for odd parity). This function returns the correct parity for the vector of bits.

Follow these steps:

1. Determine the number of 1s in the vector.
2. If the number of 1s is even and the desired parity is even (0), return 0.
3. If the number of 1s is odd and the desired parity is odd (1), return 0.
4. If the number of 1s is even and the desired parity is odd (1), return 1.
5. If the number of 1s is odd and the desired parity is even (0), return 1.

Note that you do not need to write 4 separate **if** statements, or an **if** with 3 **elseif** sections), nor should you. I was able to use a single **if/else** statement to determine the correct return value for this function. If you are not sure how to proceed, get the function working any way at all, but then go back and see how you can shorten the **if** statement. If you find a clever way to solve it and it works correctly all the time, then it is correct. Observe the relationship between the number of 1s (even or odd) and the desired parity.

```
>> parityOf([1 0 1],0)
ans =
  0
>> parityOf([1 0 1],1)
ans =
  1
```

6.5. Function appendParity

The parity bit is appended to the vector of bits.

This function has two parameters: a vector of bits, and a desired parity. Call the **parityOf** function and create a new vector by appending the returned parity bit to the end of the vector of bits. Return the new vector.

```
>> appendParity([1 0 1],0)
ans =
  1  0  1  0
>> appendParity([1 0 1],1)
ans =
  1  0  1  1
```

The body of this function should be short. *Really* short. I did it in 1 line. It's ok if you do it in 2 lines, or *maybe* even 3 lines. If you have any more lines than that, you may not understand what you are doing.

6.6. Function reseedRand

The next function you write after this one (it will be called **addNoise**) will use random numbers. But as we saw in lecture, the sequence of random numbers is the same every time you start Matlab. To overcome this problem it is necessary to re-seed the random number generator to cause it to generate a different sequence of random numbers.

Create a function called **reseedRand** that has no parameters and returns no value. Copy and paste these statements into the body of the function.

```
rng('shuffle')
```

Seriously, that's it.

If you have time you can read up on this, but you won't see this on the exam.

6.7. Function addNoise

This function simulates transmitting and subsequently receiving some bits over a noisy communications channel. Noise will cause any signal to degrade, but in a digital signal (which is just voltage levels), its effect is usually to cause some of the 1s to look like 0s, and 0s to look like 1s.

This function has two parameters: a vector of bits, and an error probability. This function iterates over the bits in the vector, and for each bit, chooses with some low probability to flip the bit (use the rand function here). For example, if the error probability is 0.5 (50%), then about half the bits will be flipped from 0 to 1 or 1 to 0. If the probability is 0.1, then about 1 out of every 10 bits will be flipped. If the error probability is 0.001, then if you give the function 1000 bits, there's a good chance that one of them will be flipped. But remember that since you are using **rand**, it is probabilistic: it's possible that 2 out of those 1000 bits will be flipped, or even 3, or maybe none. It works like this:

```
for ___
  if random number is less than the error probability
    flip the bit in the bit vector
  end
end
```

This function must return two values: the new vector that contains (maybe) some flipped bits, and the number of bits that were flipped. You need to remember how to return multiple values from a function. Inside this function you can just modify the bits in the input vector (which you were probably going to do, anyway), since it's already a copy of the argument vector. You don't need to create a new vector.

Note that a real communications channel (like a wireless network) will not be so generous as to tell us how many bits it has flipped. However, knowing the number of flipped bits lets us test, verify, and improve our programs in the presence of *simulated* noise so that we can be confident that the program will work correctly in the presence of *real* noise (this will be the Arduino project, for those choosing to do it).

```
>> char2bin('0')
ans =
  0  0  1  1  0  0  0  0
>> [b n] = addNoise(ans, 0.2)
b =
  0  0  1  0  1  0  0  1
n =
  3
```

With an error probability of 0.2 we would expect 2/10 bits to be flipped, but in this example 3/8 bits were flipped. This is still correct: it's the nature of probability.

6.8. Function checkParity

After receiving a data from a sender (or in our programs: after adding noise), the data must be checked for errors.

This function takes a vector of bits (some of which could possibly have been flipped) and a desired parity (0 for even, 1 for odd). The function returns **true** if the parity "checks out", meaning that no errors were found in the bits, or **false** if an error was found.

The vector given to this function will have 9 bits: the first 8 are the original data, and the 9th is the parity bit. Take the first 8 bits and calculate what the parity of *only those 8 bits* ought to be. Use the **parityOf** function to do this, and use the same even or odd parity value that you used to calculate the parity originally. [Note that the sender and receiver must agree to use the same parity, even or odd, in order for parity to work.] If the received parity bit (bit 9 of the group of 9 bits) is the same as the newly calculated parity bit, then it can be assumed that no error occurred during transmission. If the two parity bits are not the same, then you have determined that an error occurred during transmission.

```
>> char2bin('X')
ans =
  0  1  0  1  1  0  0  0
>> appendParity(ans,0)
ans =
  0  1  0  1  1  0  0  0  1
>> addNoise(ans, 0.1)
ans =
  0  1  0  1  1  0  1  0  1
```

(Note that your answer for **addNoise** may be different, but when I did it here a single bit was flipped.)

```
>> checkParity(ans,0)
ans =
  0
```

(0 = false, the parity is not correct.)

6.9. Script p7

Put a comment block at the beginning of this script file just like you did for the script files for your other projects.

This script should thread all these functions together, plus a few extra operations:

1. Create a variable that contains the error probability: set it to 0.1.
2. Create a variable that contains the desired parity: set it to 0 (for even).
3. Re-seed the random number generator.
4. Get a character from the user.

---- Nothing should be displayed in steps 5 through 8. ----

5. Convert the character to a vector of bits (there should be 8 bits now).
6. Append a parity bit to the bits (so 8 bits become 9 bits). These are the "transmitted" bits.
7. Add noise to the 9 bits. These are the "received" bits.
8. Check the parity of the "received" noisy bits.
9. Display the original 9 bits with a descriptive message.
10. Display the number of flipped bits with a descriptive message.
11. Display the noisy 9 bits with a descriptive message (note that they may be the same as the original bits if no bits were flipped).
12. Display the string 'no error detected' or 'error detected' depending on whether or not an error was detected.
13. Convert the first 8 bits of the "noisy" vector into a character and display it with a descriptive message.

Here's how mine runs:

```
>> p5
Enter a character: X
transmitted bit vector
  0  1  0  1  1  0  0  0  1
flipped bits
  1
received bit vector
  1  1  0  1  1  0  0  0  1
error detected
received character
Ø
```

In this first test run, there was one flipped bit. The resulting character is 11011000 in binary (without the final parity bit), or ASCII code 216, which is the empty-set character, which looks a lot like a crossed zero. Let's run it again:

```
>> p5
Enter a character: X
transmitted bit vector
  0  1  0  1  1  0  0  0  1
flipped bits
  1
received bit vector
  0  1  0  1  1  0  0  0  0
error detected
received character
X
```

In this next test run there was a single flipped bit, but it was the parity bit. The first 8 bits are unaffected, so the received character is the same as the transmitted character. Unfortunately, the receiver has no way to know that the parity bit was the one that was flipped. All it knows is that there is an error somewhere in the 9 bits.

One more run:

```
>> p5
Enter a character: X
transmitted bit vector
  0  1  0  1  1  0  0  0  1
flipped bits
  2
received bit vector
  1  1  0  1  0  0  0  0  1
no errors detected
received character
Ð
```

In this last test run there are two flipped bits and the received character is not the same as the character that the user entered, but the **checkParity** function indicates that there are no errors. Why is that? Determine the answer to this question. The key to the answer is in the **checkParity** function, but you may need to use an external resource like Google or Wikipedia to do some research to find the answer. This question is not worth points in this assignment, but there is a good chance you'll need to know this on the final exam.

7. Report

Create a Microsoft Word or OpenOffice Word file (or some other format that your TA agrees on – ask him or her if you are not sure). Save the file with the name **Project2** with a .doc or .docx format.

At the beginning of the document include this information:

Error Detection
CSE1010 Project ___, Fall 2012
Your name goes here
The current date goes here
TA: *Your TA's name goes here*
Section: *Your section number goes here*
Instructor: Jeffrey A. Meunier

Be sure to replace the parts that are underlined above.

Now create the following five sections in your document.

1. Introduction
In this section copy & paste the text from the introduction section of this assignment. (It's not plagiarism if you have permission to copy something. I give you permission.)
2. Inputs & outputs
Describe what kind of values the user is expected to supply, and what kind of values the program displays. Don't give examples, just describe the information: pretend you're telling your mom or dad or S.O. about it over the phone. "The user must enter a character, and the program displays the binary value of the character, etc." Also describe any constants used in the program (this is the error probability and the parity used, even or odd).
3. Sample run
Copy & paste two sample runs, one that shows no transmission error, and one that shows a single transmission error.
4. Source code
Copy & paste the contents of your .m files here. You do not need to write anything.

8. Submission
Submit the following things things on HuskyCT:
1. All the 8 functions and 1 script file for this project.
2. The MS Word document.

If for some reason you are not able to submit your files on HuskyCT, email your TA before the deadline. Attach your files to the email.

9. Notes
Here are some notes about working on this project:
- I can't think of anything right now.

10. Grading Rubric
Your TA will grade your assignment based on these criteria:
- (12 points) The program works correctly
- (4 points) The program is formatted neatly and correctly, and the comments are as they should be: in each function and in the script file.
- (4 points) The document contains all the correct information and is formatted neatly.

11. Getting help
Start your project early, because you will probably not be able to get help in the last few hours before the project is due.

- If you need help, send e-mail to your TA immediately.
- Go to the office hours for (preferably) your TA or any other TA. I suggest you seeing your own TA first because your TA will know you better. However, don't let that stop you from seeing any TA for help.
- Send e-mail to Jeff.
- Go to Jeff's office hours.