N-Body Simulation

CSE1010 Project 7, Fall 2012

Date: 11/12/2012

Name: William Dickson

Section: 009L

TA: Levon Nazaryan

Instructor: Jeffrey A. Meunier

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

1. **Inputs & outputs**

The only value that the user is expected to supply is the character they wanted ‘sent’. This program mimics a transmission of a character that may or may not have some error during transmission. The character is then converted into binary and is displayed as the “Original transmitted bit vector:” The program then tells you how many of the bits (0s or 1s) have been flipped (at the probability of .1 that there will be an error in each bit) and displays the new bit vector with the errorous bits as the “Received bit vector:” Whether or not there was an error detected will be displayed and then the received character based on the “Received bit vector:” will be displayed (converted back from binary). To determine whether or not there was an error the program uses something called parity. Using parity the program will create a way to check the ending solution with the beginning value. The parity is defined by the binary value of the character which is a string of 0s and 1s. The number of 1s in the binary value with either be odd or even. In this program we were looking for an odd number of 1s and 0s so if there is currently an even number of 1s then a 1 will be appended to the end of the vector to make the vector odd. For an odd number of 1s a 0 will be added. After transmission the program checks the number of 1s compared to the value on the end and if it still is an odd number of 1s then it passes the parity test.

1. **Sample Run**

**First Run**

Enter a character: i

Original transmitted bit vector:

0 1 1 0 1 0 0 1 1

Number of flipped bits: 1

Received bit vector:

0 1 1 0 1 0 1 1 1

Error Detected

Received Character: k

**Second Run**

Enter a character: i

Original transmitted bit vector:

0 1 1 0 1 0 0 1 1

Number of flipped bits: 0

Received bit vector:

0 1 1 0 1 0 0 1 1

No Errors Detected

Received Character: i

1. **Source Code**

% Error Detection

% CSE1010 Project 7, Fall 2012

% William Dickson

% 11/07/2012

% TA: Levon Nazaryan

% Section: 009L

% Instructor: Jeffrey A. Meunier

clc % clear command window

clear % clear all variables

err\_prob = 0.1;

desired\_par = 1;

reseedRand

c = getChar;

b = char2bin(c);

b\_appended = appendParity(b,desired\_par);

[b\_noise,n] = addNoise(b\_appended,err\_prob);

check = checkParity(b\_noise,desired\_par);

disp('Original transmitted bit vector:')

disp(b\_appended)

fprintf('Number of flipped bits: %g\n',n)

disp('Received bit vector:')

disp(b\_noise)

if check == 0

disp('Error Detected')

else

disp('No Errors Detected')

end

fprintf('Received Character: %c\n',bin2char(b\_noise(1:8)))

function [b,n] = addNoise(b,prob)

% Given a vector of numbers and a probability of error (on each element of

% the vector) this function will randomly flip a 0 to a 1 or vise versa at

% the given probability.

% Use: addNoise(b,prob)

n = 0;

for i=1:length(b)

if rand < prob

if b(i) == 0

b(i) = 1;

else

b(i) = 0;

end

n = n + 1;

end

end

end

function b = appendParity(b,parity\_bit)

% Given a vector of numbers and a parity value (0 or 1) this function will

% add a value to the end of the vector to make the vector match the parity

% Use: appendParity(b,parity\_bit)

b(end+1) = parityOf(b,parity\_bit);

end

function c = bin2char(b)

% Given a vector of binary numbers this will convert the vector to the

% corresponding ASCII character defined by them.

% Use: bin2char(b)

b\_str = char(b+48);

c = char(bin2dec(b\_str));

end

function b = char2bin(c)

% Given a character this will convert the vector to the

% corresponding ASCII number and then the binary number that it corresponds to.

% Use: char2bin(c)

bin\_str = dec2bin(c);

bin\_str\_long = sprintf('%08s',bin\_str);

b\_vec = bin\_str\_long - 48;

b = logical(b\_vec);

end

function check = checkParity(b\_parity,parity)

% Given a vector of binary numbers and a parity value this function will

% check to see if the parities are equal. If they are equal it returns a 1

% if not then a 0.

% Use: checkParity(b\_parity,parity)

if parity == parityOf(b\_parity(1:end-1),b\_parity(end))

check = 1;

else

check = 0;

end

end

function c = getChar

% This function prompts the user for a character and if the character

% string is more than one character long then it will truncate the string

% to the first character.

% Use: getChar

c = input('Enter a character: ', 's');

c = c(1);

end

function parity\_bit = parityOf(b,desired\_par)

% This function determines the parity of a vector and if it is equal to the

% desired parity then it sets the 'parity\_bit' equal to 0, otherwise it

% sets it eqaul to 1.

% Use: parityOf(b,desired\_par)

if rem(length(find(b)),2) == desired\_par

parity\_bit = 0;

else

parity\_bit = 1;

end

end

function reseedRand

% This function resets the random number generator to make a 'more' random

% number.

% Use: reseedRand

rng('shuffle')

end