HW 8

Question 5

Checking Portion with Moser Example

Code

```
% Question 5, Check

% Design binary arithmetic code for ternary random variable

% We will code to encode a single sequence (given in Moser) to check code
% correctness before we perform the tasks underlying the problem


% Block Length, M = Sequence length to be encoded

% U = {a,b,c} ternary RV, alpha_1 = a
%                         alpha_2 = b
%                         alpha_3 = c

% P = {p_a,p_b,p_c}     , p_a = p_alpha_1 = 0.5
%                         p_b = p_alpha_2 = 0.3
%                         p_c = p_alpha_3 = 0.2

% The above PMF and ordering is as given in Moser Example

% a ------- alpha_1 ------- 1           (representation)
% b ------- alpha_2 ------- 2           (representation)
% c ------- alpha_3 ------- 3           (representation)

% M will be decided from the sequence length, which is to be encoded
% U and PMF given
% Thus encoding algorithm given in Moser can be performed



% NOTE
% We will consider the sequence to be described in terms of 1,2 and 3's
% instead of a, b and c's. We could write extra code to convert the a, b
```

```
% and c's into 1,2 and 3's but this is trivial, and seems unnecessary. Thus
% a sequence such as abcca will correspond to input 12331, and we will
% presume that this is the input the code will be given, to further encode
% it.


% sequence given in Moser = baabcabbba = 2112312221

seq = [2 1 1 2 3 1 2 2 2 1];

pmf = [0.5 0.3 0.2];          % pmf as given in Moser example

D = 2;

% Additional Note
% U = {a,b,c} = {alpha_1,alpha_2,alpha_3}
% pmf = {P(alpha_1),P(alpha_2),P(alpha_3)} = {P(a),P(b),P(c)}
%                                          = {0.5,0.3,0.2}

% Also note a,b,c and 1,2,3 are one and the same

% Arrangement of a,b,c to alpha's doesn't matter, but once fixed, should
% stay consistent to ensure algorithm sanity

% Note : r in alpha_1, alpha_2,....,alpha_r here is equal to 3
% r = 3

% alpha_1 = a = 1
% alpha_2 = b = 2
% alpha_3 = c = 3

r = size(pmf,2);              % alphabet size = non-zero probabilities given
```

```
M = size(seq,2);                % Block length = sequence length here

f = zeros(1,r);

for i = 2:r
    f(1,i) = sum(pmf(1,1:i-1),2);
end

% pmf = {p(alpha_1),p(alpha_2),p(alpha_3)}
% f = {f(alpha_1),f(alpha_2),f(alpha_3)}
% f(alpha_1) = 0
% f(alpha_c) = sum pmf entries from 1 to c-1, c > 1
% Since we know that f(alpha_r) = sum p(alpha_i) over i = 1 to r-1, r > 1


p = 1;
F = 0;

for k = 1:M
    obs = seq(1,k);
    F = F + p*f(1,obs);                 % since observation obs = z corres
    p = p*pmf(1,obs);
end

% F has F_M and p has p_M

len = ceil((log(1/p))/(log(D))) + 1;

F_whole_dec = F;    % entirety of F, in decimal, untruncated

max_length = 100;

F_vec_bin = zeros(1,max_length);                % Two things assumed, 1. Th
```

```
                                                        %                  2. Th



% F_vec_bin is a row vector which contains the post-decimal point binary rep
% We assume that the scope of this exercise is not general, and, the limit
% of 100 on the maximum description length of the binary representation of
% F_whole_dec is more than sufficient.

% To code a more general case, with a possibly lengthy code, much, much
% lengthy, one could create a different data structure in MATLAB, one that
% doesn't require size to be initiated, and then append onto that
% structure. We can then, after a certain length, end the conversion
% program if the length exceeds beyond a certain length, length that is
% application based.


% Thus in our assumption both natural binary description length and length
% obtained must be <= 100. This can be changed based on application by
% changing the max_length here, thus we can argue that the code is general
% enough for the current exercise purpose.

% Converting F_whole_dec into binary now

% Note F_whole_dec will contain the actual untruncated decimal value, while
% F which till this point is equal to F_whole_dec, will be used and
% modified to obtain its binary representation

% Note : Naturally, F >= 0

if F >= 1
    disp('Some error in algorithm, making untruncated F in decimal greater t
    flag = 1;
```

```
elseif F > 0        % F_whole_dec is in (0,1). We now convert this in binary.
    length_flag = 0;
    flag = 2;
    for j = 1:100
        F = 2*F;
        if F < 1
            F_vec_bin(1,j) = 0;
        elseif F > 1
            F_vec_bin(1,j) = 1;
            F = F - 1;
        else
            disp('The binary representation of untruncated decimal codeword
            F_vec_bin(1,j) = 1;
            if j ~= 100
                disp('The length is less than 100, of the representation')
                F_vec_bin(1,j+1) = NaN;     % to represent termination
                length_flag = 1;
            else
                disp('The length is exactly 100')
                length_flag = 2;
            end
            break
        end
    end
else
    % Implies F = 0 (or F_whole_dec = 0)
    % F_vec_bin stays filled with zeros, no change
    flag = 0;
end



% CASES
```

```
% flag = 1
% some error, F >= 1, we have a weird situation

% flag = 0
% F = 0. Thus F_vec_bin stays zero, which is correct

% flag = 2
% F is in (0,1)
%       length_flag = 0
%             implies that full binary representation has length > 100
%       length_flag = 1
%             implies this representation is of length < 100
%             ending bit comes before NaN in our F_vec_bin
%       length_flag = 2
%             implies the representation has length exactly 100

% NOTE :
% F_whole_dec contains now the untruncated decimal representation of our
% codeword as F may have been modified to obtain the binary representation
% in this case


if flag == 1
    disp('Some weird error')
elseif flag == 0

    F_bin = 0;                                       % F_whole_decimal is zero i

    if len > 100
        F_bin_trunc = zeros(1,len);                  % truncated binary code
        to_add = zeros(1,len);
        to_add(1,len) = 1;
```

```
            F_bin_final = to_add + F_bin_trunc;      % direct addition works sin
        else
            F_bin_trunc = F_vec_bin(1,1:len);        % as is, since flag is 0, F
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = to_add + F_bin_trunc;      % direct possible due to F_
        end

    else

        if (length_flag == 0 && len > 100)

            F_bin = F_vec_bin;
            disp('Representation incomplete due to maximum length = 100 assumpti
            F_bin_trunc = zeros(1,len);
            F_bin_trunc(1,1:100) = F_vec_bin(1,1:100);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,'left

        elseif (length_flag == 0 && len <= 100)

            F_bin = F_vec_bin;
            F_bin_trunc = F_vec_bin(1,1:len);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,'left


        % For complete representations within length 100 (including 100) of the
        % binary code, we presume that the system is stable and thus len in
        % that case will be equal to or smaller than the total representational
        % length of the binary code obtained from the untruncated decimal code
```

```
        elseif (length_flag == 1)

            [max_val_red,term_indx] = max(isnan(F_vec_bin));
            F_bin = F_vec_bin(1,1:(term_indx-1));
            F_bin_trunc = F_bin(1,1:len);         % stable system, full represe
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,'lef

        else

            % length_flag is 2

            F_bin = F_vec_bin;              % actual representation ends exactly
            F_bin_trunc = F_bin(1,1:len);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,'lef

        end



    end


    if flag == 1
        disp('There is nothing to be displayed')
    else
        format long
        disp('The various results, with corresponding cases and explanations in
        disp('The code word, untruncated, in decimal is:')
        disp(F_whole_dec)
        disp('The code word, untruncated, in binary is:')
        disp(F_bin)
        disp('The code word, truncated, in binary is:')
        disp(F_bin_trunc)
        disp('The final code word, in binary, post addition of 2^-l term, is:')
        disp(F_bin_final)
    end

```

The code has been explained in detail in the various comments.

This code takes care of a huge range of possibly unsavory situations, due to which the code length increased a little bit, despite the simplicity of the arithmetic coding algorithm.

When run, the results agreed with those given in Moser, markedly.

Results of the Check Code

```
>> HW8_5_Check
The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100
The length is less than 100, of the representation
The various results, with corresponding cases and explanations in the comments of the code, go thus
The code word, untruncated, in decimal is:
   0.557063750000000

The code word, untruncated, in binary is:
  Columns 1 through 31

    1     0     0     0     1     1     1     0     1     0     0     1     1     0     1     1     1     0     1     1     1     0     1     0     1     1     0     1     1     1     0

  Columns 32 through 53

    0     0     0     0     0     1     0     0     1     1     0     0     0     0     0     0     0     1     0     1     1     1

The code word, truncated, in binary is:
    1     0     0     0     1     1     1     0     1     0     0     1     1     0     1     1     1

The final code word, in binary, post addition of 2^-l term, is:
    1     0     0     0     1     1     1     0     1     0     0     1     1     1     0     0     0
```

Above snap is unclear, so we copy paste the result tab.

>> HW8_5_Check

The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100

The length is less than 100, of the representation

The various results, with corresponding cases and explanations in the comments of the code, go thus

The code word, untruncated, in decimal is:

0.557063750000000

The code word, untruncated, in binary is:

Columns 1 through 31

1 0 0 0 1 1 1 0 1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 1 1 0

Columns 32 through 53

0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1

The code word, truncated, in binary is:

1 0 0 0 1 1 1 0 1 0 0 1 1 0 1 1 1

The final code word, in binary, post addition of 2^-l term, is:

1 0 0 0 1 1 1 0 1 0 0 1 1 1 0 0 0

As can be seen from Moser text, the final code word obtained via the code and via Moser is the same, proving that the code works correctly.

SUB-PARTS a and b for all sequence u1, u2 and u3

Code

The coding of all results done together. Answer to the question in Part b (the latter part, relating to equations in Moser) and discussion Part c done later

```
% Question 5 | All parts for all codes, excluding the c) discussion and the
% answer to question asked in latter part of b)

% Sub-part a) and b) for all the three codewords will be done
% simultaneously. For each code, encoding, and the various descriptions in
% a) and b) as described, will all, be done in this single code.

% Once we obtain results we will attempt to:
% 1. Answer the question asked in b) (the latter part, the code results
% will be obtained here itself)
% 2. Discuss the results, as asked, in c)

% All comments have been removed since in the check section their
% explanatory purpose has been fulfilled, and this makes navigating code
% easier


% The pmf will remain same, and, constant, for all sequences





seq1 = [1 3 1 2 3 2 1 2];          % u1

seq2 = [1 3 1 2 3 2 1 3];          % u2

seq3 = [1 3 1 2 3 2 2 2];          % u3

seq_cell = cell(1,3);

seq_cell{1,1} = seq1;
```

```
seq_cell{1,2} = seq2;
seq_cell{1,3} = seq3;

for ite = 1:3

    seq = seq_cell{1,ite};
    pmf = [0.5 0.3 0.2];            % stays same for all

    D = 2;

    r = size(pmf,2);
    M = size(seq,2);
    f = zeros(1,r);

    for i = 2:r
        f(1,i) = sum(pmf(1,1:i-1),2);
    end



    p = 1;
    F = 0;

    for k = 1:M
        obs = seq(1,k);
        F = F + p*f(1,obs);
        p = p*pmf(1,obs);
    end

    len = ceil((log(1/p))/(log(D))) + 1;                            % We

    F_whole_dec = F;
```

```
max_length = 100;

F_vec_bin = zeros(1,max_length);



if F >= 1
    disp('Some error in algorithm, making untruncated F in decimal grea
    flag = 1;
elseif F > 0
    length_flag = 0;
    flag = 2;
    for j = 1:100
        F = 2*F;
        if F < 1
            F_vec_bin(1,j) = 0;
        elseif F > 1
            F_vec_bin(1,j) = 1;
            F = F - 1;
        else
            disp('The binary representation of untruncated decimal code
            F_vec_bin(1,j) = 1;
            if j ~= 100
                disp('The length is less than 100, of the representatio
                F_vec_bin(1,j+1) = NaN;
                length_flag = 1;
            else
                disp('The length is exactly 100')
                length_flag = 2;
            end
            break
```

```
            end
        end
    else
        flag = 0;
    end



    if flag == 1
        disp('Some weird error')
    elseif flag == 0

        F_bin = 0;

        if len > 100
            F_bin_trunc = zeros(1,len);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = to_add + F_bin_trunc;
        else
            F_bin_trunc = F_vec_bin(1,1:len);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = to_add + F_bin_trunc;
        end

    else

        if (length_flag == 0 && len > 100)

            F_bin = F_vec_bin;
            disp('Representation incomplete due to maximum length = 100 ass
```

```
            F_bin_trunc = zeros(1,len);
            F_bin_trunc(1,1:100) = F_vec_bin(1,1:100);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,

        elseif (length_flag == 0 && len <= 100)

            F_bin = F_vec_bin;
            F_bin_trunc = F_vec_bin(1,1:len);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,




        elseif (length_flag == 1)

            [max_val_red,term_indx] = max(isnan(F_vec_bin));
            F_bin = F_vec_bin(1,1:(term_indx-1));
            F_bin_trunc = F_bin(1,1:len);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,

        else

            F_bin = F_vec_bin;
            F_bin_trunc = F_bin(1,1:len);
            to_add = zeros(1,len);
            to_add(1,len) = 1;
```

```
            F_bin_final = de2bi(bi2de(F_bin_trunc,'left-msb')+bi2de(to_add,'

        end



    end

    if flag ~= 1
        len_dec = ceil((log(1/p))/(log(10))) + 1;
        F_trunc_dec_interim = F_whole_dec*(10^len_dec);
        F_trunc_dec = floor(F_trunc_dec_interim);
        F_extr_trunc_dec_interim = F_whole_dec*(10^(len_dec-1));
        F_extr_trunc_dec = floor(F_extr_trunc_dec_interim);
        F_final_dec = (F_trunc_dec*(10^-len_dec) + (10^-len_dec))*(10^len_de
        F_final_dec = cast(F_final_dec,'int32');
        F_trunc_dec = cast(F_trunc_dec,'int32');
        F_extr_trunc_dec = cast(F_extr_trunc_dec,'int32');
    end


    if flag == 1
        disp('There is nothing to be displayed')
    else
        format long
        disp('The results for SEQUENCE:')
        disp(ite)
        disp('The various results, with corresponding cases and explanations
        disp('The codeword length in binary is:')
        disp(len)
        disp('The codeword length in decimal, base 10 is:')
        disp(len_dec)
        disp('The code word, untruncated, in decimal is:')
        disp(F_whole_dec)
        disp('The code word, truncated, in decimal, without adding 2^-1 term
        disp(F_trunc_dec)
        disp('The code word, truncated, in decimal, with the addition of 2^-
        disp(F_final_dec)
        disp('The code word, truncated to the l-1 length (smaller representa
        disp(F_extr_trunc_dec)
        disp('The code word, untruncated, in binary is:')
        disp(F_bin)
        disp('The code word, truncated, in binary is:')
        disp(F_bin_trunc)
        disp('The final code word, in binary, post addition of 2^-1 term, is
        disp(F_bin_final)

        disp('NOTE : Decimal representations are converted to codewords afte
    end

end
```

Result Snap

```
>> HW8_5_a_b_for_all
The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100
The length is less than 100, of the representation
The results for SEQUENCE:
     1

The various results, with corresponding cases and explanations in the comments of the code, go thus:
The codeword length in binary is:
    14

The codeword length in decimal, base 10 is:
     5

The code word, untruncated, in decimal is:
   0.438725000000000

The code word, truncated, in decimal, without adding 2^-1 term, is:
   43872

The code word, truncated, in decimal, with the addition of 2^-1 (D^-1 actually, 10^-1 here, since D = 10) term, is:
   43873

The code word, truncated to the l-1 length (smaller representation), in decimal, obviously without any term addition, is:
   4387

The code word, untruncated, in binary is:
  Columns 1 through 31

     0     1     1     1     0     0     0     0     0     1     0     1     0     0     0     0     0     1     0     0     1     0     0     0     0     0     0     1     0     1     1

  Columns 32 through 52

     0     1     1     1     1     0     0     0     0     0     0     0     0     0     1     1     0     1     0     0     1

The code word, truncated, in binary is:
     0     1     1     1     0     0     0     0     0     1     0     1     0     0

The final code word, in binary, post addition of 2^-1 term, is:
     1     1     1     0     0     0     0     0     1     0     1     0     1

NOTE : Decimal representations are converted to codewords after removing the decimal point, only exception is for the true, untruncated, whole, decimal representation
The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100
The length is less than 100, of the representation
The results for SEQUENCE:
     2

The various results, with corresponding cases and explanations in the comments of the code, go thus:
The codeword length in binary is:
    15

The codeword length in decimal, base 10 is:
     6

The code word, untruncated, in decimal is:
   0.438860000000000

The code word, truncated, in decimal, without adding 2^-1 term, is:
   438860

The code word, truncated, in decimal, with the addition of 2^-1 (D^-1 actually, 10^-1 here, since D = 10) term, is:
   438861

The code word, truncated to the l-1 length (smaller representation), in decimal, obviously without any term addition, is:
   43886


The code word, untruncated, in binary is:
  Columns 1 through 31

     0     1     1     1     0     0     0     0     0     1     0     1     1     0     0     1     0     0     1     0     0     0     0     1     0     0     0     0     0     0     1

  Columns 32 through 54

     1     1     0     0     0     0     1     0     1     1     1     0     0     0     1     1     0     1     0     0     0     0     1

The code word, truncated, in binary is:
     0     1     1     1     0     0     0     0     0     1     0     1     1     0     0

The final code word, in binary, post addition of 2^-1 term, is:
     1     1     1     0     0     0     0     0     1     0     1     1     0     1

NOTE : Decimal representations are converted to codewords after removing the decimal point, only exception is for the true, untruncated, whole, decimal representation
The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100
The length is less than 100, of the representation
The results for SEQUENCE:
     3

The various results, with corresponding cases and explanations in the comments of the code, go thus:
The codeword length in binary is:
    15

The codeword length in decimal, base 10 is:
     6

The code word, untruncated, in decimal is:
   0.439085000000000
```

```
The codeword length in decimal, base 10 is:
     6

The code word, untruncated, in decimal is:
   0.439085000000000

The code word, truncated, in decimal, without adding 2^-1 term, is:
   439085

The code word, truncated, in decimal, with the addition of 2^-1 (D^-1 actually, 10^-1 here, since D = 10) term, is:
   439086

The code word, truncated to the l-1 length (smaller representation), in decimal, obviously without any term addition, is:
   43908

The code word, untruncated, in binary is:
  Columns 1 through 31

     0     1     1     1     0     0     0     0     0     1     1     0     0     1     1     1     1     1     0     1     1     1     1     1     1     1     1     0     0     0     1

  Columns 32 through 53

     1     0     0     1     0     1     0     1     0     0     0     0     0     0     1     1     0     0     1     1     0     1

The code word, truncated, in binary is:
     0     1     1     1     0     0     0     0     0     1     1     0     0     1     1

The final code word, in binary, post addition of 2^-1 term, is:
     1     1     1     0     0     0     0     0     1     1     0     1     0     0

NOTE : Decimal representations are converted to codewords after removing the decimal point, only exception is for the true, untruncated, whole, decimal representation
```

Again, due to the length and width of the results, they can’t be clearly seen in the Result Snaps, so we have copy pasted the entire MATLAB Results Tab to summarize the results

>> HW8_5_a_b_for_all

The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100

The length is less than 100, of the representation

The results for SEQUENCE:

    1

The various results, with corresponding cases and explanations in the comments of the code, go thus:

The codeword length in binary is:

   14

The codeword length in decimal, base 10 is:

    5

The code word, untruncated, in decimal is:

   0.438725000000000

The code word, truncated, in decimal, without adding 2^-1 term, is:

43872

The code word, truncated, in decimal, with the addition of 2^-l (D^-l actually, 10^-l here, since D = 10) term, is:

43873

The code word, truncated to the l-1 length (smaller representation), in decimal, obviously without any term addition, is:

4387

The code word, untruncated, in binary is:

Columns 1 through 31

0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1

Columns 32 through 52

0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1

The code word, truncated, in binary is:

0 1 1 1 0 0 0 0 0 1 0 1 0 0

The final code word, in binary, post addition of 2^-l term, is:

1 1 1 0 0 0 0 0 1 0 1 0 1

NOTE : Decimal representations are converted to codewords after removing the decimal point, only exception is for the true, untruncated, whole, decimal representation

The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100

The length is less than 100, of the representation

The results for SEQUENCE:

2

The various results, with corresponding cases and explanations in the comments of the code, go thus:

The codeword length in binary is:

15

The codeword length in decimal, base 10 is:

6

The code word, untruncated, in decimal is:

0.438860000000000

The code word, truncated, in decimal, without adding 2^-1 term, is:

438860

The code word, truncated, in decimal, with the addition of 2^-l (D^-l actually, 10^-l here, since D = 10) term, is:

438861

The code word, truncated to the l-1 length (smaller representation), in decimal, obviously without any term addition, is:

43886

The code word, untruncated, in binary is:

Columns 1 through 31

0 1 1 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0
0 0 0 0 0 1

Columns 32 through 54

1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1

The code word, truncated, in binary is:

0 1 1 1 0 0 0 0 0 1 0 1 1 0 0

The final code word, in binary, post addition of 2^-l term, is:

1 1 1 0 0 0 0 0 1 0 1 1 0 1

NOTE : Decimal representations are converted to codewords after removing the decimal point, only exception is for the true, untruncated, whole, decimal representation

The binary representation of untruncated decimal codeword has length, post-decimal point, less than or equal to 100

The length is less than 100, of the representation

The results for SEQUENCE:

3

The various results, with corresponding cases and explanations in the comments of the code, go thus:

The codeword length in binary is:

15

The codeword length in decimal, base 10 is:

6

The code word, untruncated, in decimal is:

0.439085000000000

The code word, truncated, in decimal, without adding 2^-1 term, is:

439085

The code word, truncated, in decimal, with the addition of 2^-l (D^-l actually, 10^-l here, since D = 10) term, is:

439086

The code word, truncated to the l-1 length (smaller representation), in decimal, obviously without any term addition, is:

43908

The code word, untruncated, in binary is:

Columns 1 through 31

0 1 1 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1
1 1 0 0 0 1

Columns 32 through 53

1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1

The code word, truncated, in binary is:

0 1 1 1 0 0 0 0 0 1 1 0 0 1 1

The final code word, in binary, post addition of 2^-l term, is:

1 1 1 0 0 0 0 0 1 1 0 1 0 0

NOTE : Decimal representations are converted to codewords after removing the decimal point, only exception is for the true, untruncated, whole, decimal representation

>>

The results are all well summarized and described above, including all the details asked and more, and can be referred to. These results are self-explanatory.

PART b QUESTIONS

COMPARISON OF VARIOUS DECIMAL REPRESENTATIONS

IMPACT OF 5.23 AND 5.24 MOSER EQUATIONS

CAN CODEWORD FALL OUT OF THE DESIRED INTERVAL OF THE CDF?

Answer:

The reason why 2^-l is added and length made 1 more than the Shannon length is to ensure that the codes remain prefix-free, since, the PMF is not ordered.

The unordering of the PMF corresponds to a situation whereby Arithmetic Coding could give codes that are not prefix-free, if, we used the original Shannon Length and didn't add 2^-l.

This situation can be also seen from the above sequences and their results, corresponding to representations where we have 2^-l not added to the correctly truncated sequences and where we have truncated the sequences to a length equal to Shannon Length (1 less than normal Arithmetic Coding length).

Since prefix-free nature is no longer guaranteed when we truncate to a shorter length (Shannon Length) or when we don't add 2^-l to the sequence correctly truncated, the codeword may fall out of the desired cdf interval making the code not prefix-free.

Thus truncating to one more than shannon length and adding 2^-l both ensure that the arithmetic code remains prefix-free, and within correct cdf range.

PART c

RESULT DISCUSSION

Answer:

We have already discussed things in detail, as such, since the beginning, but in summary, we obtained Arithmetic Codes, both binary and decimal for different versions (truncated, untruncated, incorrectly truncated, correctly truncated without 2^-l added) and we saw how it affects the arithmetic code for different sequences for a ternary random variable.

We can say that truncating to a length more than Shannon Length and adding 2^-l is very important for the Arithmetic Codes, since doing so ensures that code stays prefix-free.