Danmarks Tekniske Universitet

01259 Error Correcting Codes

Project 2

Submitted by:

Sameer K.C. (s094746)

Fariha Nazmul (s094747)

Amandeep Dhir (s094744)

Project:

We have to write a decoding program for an interesting code and evaluate the performance.

Choice of Code:

We choose a (n,k,d) Reed Solomon code over $\mathbf{F}_{256}$ as an interesting code.

In (n,k,d) code, n=255, k=239 and d=n-k+1=17. So, this code can correct up to 8 errors i.e. t=8.

Our prime is 2 i.e. p=2 and m=8 (2^8=256).

We choose the default primitive polynomial used in MATLAB for $F_{256.}$

Primitive polynomial = D^8+D^4+D^3+D^2+1 ------- (i)

Let α be the primitive element, then the generator polynomial is defined as

$g(x)= (x- \alpha)(x- \alpha^2)(x- \alpha^3)..(x- \alpha^{n-k})$ -------------- (ii)

MATLAB Representation of Galois Field:

For a Galois field the MATLAB representation is shown as below for some elements of F_{16}. The polynomial 0 is represented by –Inf power to α. We'll be using the powers of α in exponential format to denote the elements of the field.

Exponential Format	Polynomial Format	Row of MATLAB Matrix of Elements
$\alpha^{-\mathrm{Inf}}$	0	0 0 0 0
α^0	1	1 0 0 0
α^1	x	0 1 0 0
α^2	1+x	1 1 0 0

For example, if we have a received polynomial $r(x)= 1+x+\alpha^2x^3$ then we represent it in MATLAB as r = [0 0 –Inf 2]

The position in the matrix gives the power of x and the coefficients are the powers of α. It is formed by listing the coefficients of the polynomial in order of ascending powers of x.

Encoding:

We're using systematic encoding of Reed-Solomon codes.

- Information polynomial i(x) is k bits long.
- Multiplying i(x) with x^{n-k} right shifts the message to n-k place. And we can add the parity polynomial p(x) to the leftmost n-k bits.
- x^{n-k} i(x) =q(x) g(x) + p(x)
- p(x) = x^{n-k} m(x) modulo g(x)
- So our encoded codeword, C(x) = p(x) + x^{n-k} i(x)

Decoding:

We're using the Euclidean algorithm for decoding Reed-Solomon codes.

- We calculate syndromes S(i) = r(α^i), where i=1..n-k and r(x) = received polynomial.
- If all S(i) are zero (in Galois field representation it is –Inf) then the received code has no errors and we return the receive code word as the decoded code word.
- Otherwise, we calculate the syndrome polynomial S(x) by using S(x) = $_{i=1}\sum^{2t}$ S(i) x^{2t-i}
- Now, we run Euclidean algorithm on x^{2t} and S(x) until deg(r_j) < t.
- Then we calculate the roots of gj. If the number of roots is not equal to degree of g_j ,then it means that there are more errors than t.
- The roots give the error positions from which we can calculate the error polynomial e(x) by theorem 11.2.2 in the book
- The corrected codeword is then given as **c(x)=r(x)+e(x)**
- If dividing the decoded codeword by g(x) yields zero then it's a valid codeword else it's not.
- If we received the decoded codeword as the codeword that we encoded then it's we call it successful decoding.
- If we get the decoded codeword same as the codeword we sent with error patterns then no change is made to the received codeword.
- Else there is decoding error i.e. it is decoded to a wrong codeword.

Outputs and Analysis:

- We generated a zero codeword with length n as a single frame and using randerr( ) function of MATLAB we tried to create errors at j different places in a loop for 100 similar frames. Then we tried to decode each of these 100 code words.
- Here is an extract of the output with errors at 8 different places for each frame. With 8 errors, we can successfully decode all the 100 code words.
  (Here, -Inf dentotes the zero and 0 denotes the error pattern here)

Processing frame.. 5
Received word:
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf 0 -Inf -Inf -Inf -Inf -Inf -Inf 0 -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf 0 -Inf -Inf
0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf 0 -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf 0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf 0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf 0 -Inf -Inf
Decoded word:
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf

Succesful Decoding

- Now we have checked with 9 errors in each frame. The output says that it can't correct the errors because there are more than 8 errors. It is because the degree of gj is not equal to number of roots of gj. And we're returning the received code word as the decoded codeword, without making any error correction. And it also shows that the received/decoded codeword is not a valid codeword.

```
Processing frame.. 5
Received word:
-Inf -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf    0 -Inf
0 -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
  There must have occurred more than 8 errors.
Decoded word is not a valid codeword!
Decoded word:
-Inf -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf    0 -Inf
0 -Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf    0 -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
```

-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
-Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf -Inf
No Change

- We also checked for the 10 errors and the output was similar to above.
- It can also be mentioned that, with errors > t there is a probability of 1/t! that the received codeword will decode into a wrong codeword but still a valid codeword. While experimenting with different 100 frames we couldn’t find this decoding error.

MATLAB and other functions used:

MATLAB functions:

- To calculate the default irreducible polynomial for field p^m, we use **gfprimdf(m,p)**
- **gftuple()** over field 8 and prime 2 to get the get the galois field elements
- **gfadd( )**, **gfsub( ), gfdeconv( )**, **gfconv( )** to add, subtract, divide and multiply polynomials over Galois field and **gfmul( )** to multiply elements of the field.
- **randerr( )** to add random errors

User-defined functions:

- Function **generatorPolynomial( )** takes 2t and gftuple field arguments and returns a generator polynomial by using the formula (ii).
- Function **RSencoder( )** is used for encoding k bits of information in (n,k) RS code.
- Function **RSdecoder( )** is used for decoding the RS code.
- Function **gfeuclidRS( )** takes two polynomial a and b and error value t and field as argument and runs Euclid algorithm on a and b until remainder has degree less than t.
- Function **gfallroots( )** calculates roots of a given polynomial i.e. g_j.
- Function **gfdifferentiate( )** differentiates a given polynomial i.e. g_j.
- Function **gfpolyval( )** gives a value in exponential format by evaluating the polynomial at α^i over given field.
- File **evaluatePerformance.m** is used for checking the performance of the (255,239) Reed-Solomon code.
- File **testDecoder.m** checks the decoder for one code word.

Appendix

generatorPolynomial.m

```
function g = generatorPolynomial(twoT, field)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct the generator polynomial of the twoT/2 error correcting  %
% Reed Solomon code from the product of (x+alpha^i), where i=1..twoT %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Input:                                                               %
%       twoT:   if the code is t error correcting then 2*t           %
%       field:  list of all elements in the field p^m                %
%               generated using gftuple                              %
%Output:                                                              %
%       g:      the generator polynomial in matrix                   %
%               representing polynomail format                       %
%               i.e. [1 0 1 1 1 0 0 0 1]  for field 2^8              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %generate the generator polynomial

    %represents (alpha + X )
    gen = [1 0];
    temp(1) = gen(1);

    for i = 1:twoT-1
        %temp represents (alpha^i + X )
        temp(1) = gfmul(temp(1),1,field);
        temp(2) = 0;
        %muliplies ((alpha + X )...(alpha^i-1 + X )) and (alpha^i + X )
        gen = gfconv(gen,temp,field);
    end
    g = gen;

end
```

RSencoder.m

```
function code = RSencoder(info)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Encode a (255,239) Reed Solomon code using the systematic encoding  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Input:                                                               %
%       info:     information bits represented in field              %
%Output:                                                              %
%       code:     an encoded codeword in matrix representing         %
%                 polynomial format in field                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
    disp('Receiving information bits..');
    %prime
    p = 2;
    % Reed Solomon code over GF(2^m)
    m = 8; %8
    % Length of codeword
    n = 2^m -1;

    % number of errors can be corrected
    t = 8; %8;
    twoT = 2*t;

    % Dimension of codeword
    k = n - twoT; %239

    %default primitive polynomial
    prim_poly = gfprimdf(m,p);

    %generate a list of elements of GF(2^m)
    field = gftuple([-1:p^m-2]',m,p);

    %generate the generator polynomial
    g = generatorPolynomial(twoT, field);

    disp('Encoding information bits..');

    %Systematic encryption
    %parity bits are calculated by (X^(n-k).i(X)) / g(X)
    %codeword = (X^(n-k).i(X)) + parity bits

    %a polynomial representing X^(n-k)
    shiftPoly(1:n-k) = -Inf;
    shiftPoly(n-k+1) = 0;
    %multiplying it with the info to shift
    shiftInfo = gfconv(info,shiftPoly,field);

    %divide shifted info by g(x)
    [quot, parity] = gfdeconv(shiftInfo, g, field);

    %if padding is needed for the parity bits
    temp = -Inf;
    while length(parity) < n-k
        parity = [parity temp];
    end

    %concatenate the parity bits to the data
    code = [parity info];

end
```

RSdecoder.m

```
function decoded = RSdecoder(recWord)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Decode a (255,239) Reed Solomon code using the Euclidean algorithm  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Input:                                                              %
%       recWord:  a received codeword in matrix representing         %
%                 polynomial format in field                         %
%                 i.e. [1 0 2 3] for A + x + A^2x^2 + A^3x^3         %
%                 where A is primitive element in the field          %
%Output:                                                             %
%       decoded:  a decoded codeword in matrix representing          %
%                 polynomial format in field                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    disp('Receiving the code..');
    %recWord = [7 14 12 14 9 10 14 6 11 1 5 13 11 14 9];

    %prime
    p = 2;
    % Reed Solomon code over GF(2^m)
    m = 8; %8
    % Length of codeword
    n = 2^m -1;

    % number of errors can be corrected
    t = 8; %8;
    twoT = 2*t;

    % Dimension of codeword
    k = n - twoT; %239

    %default primitive polynomial
    prim_poly = gfprimdf(m,p);

    %generate a list of elements of GF(2^m)
    field = gftuple([-1:p^m-2]',m,p);

    %generate the generator polynomial
    g = generatorPolynomial(twoT, field);

    %convert any negative value to -Inf
    for i = 1:n
       if (recWord(i) < 0)
           recWord(i) = -Inf;
       end
    end


    disp('Decoding the code. Please wait...');
    %calculate syndromes
    S = [];
    %to get S(point) evaluating received polynomial at alpha^point
    for point = 1:twoT
        S(point)= gfpolyval(recWord,point,n,field);
    end
```

```
    %check if there is no error
    emptyPoly(1:twoT) = -Inf ;
    if(isequal(S, emptyPoly))
        decoded = recWord;
    else

        %generate the polynomial S(x)
        for i = 1:twoT
            Sx(i) = S(twoT-i+1);
        end

        %generate a polynomial for x^2t
        xPow2t(1: twoT)= -Inf;
        xPow2t(twoT+1) = 0;

        %run the Euclid alg on x^2t, S(x)
        [gj, rj] = gfeuclidRS(xPow2t, Sx, t, field);

        %find the roots of gj
        roots = gfallroots(gj,n,field);

        %differentiate gj
        gjDiff = gfdifferentiate(gj);

        %check the number of roots equals the degree of gj
        if not(length(roots)==(length(gj)-1))
            disp(sprintf('  There must have occurred more than %d
                      errors.',t));
            decoded = recWord;
            %return
        else
            %find the error polynomial
            e(1:n) = -Inf;
            for r = 1 : length(roots)
                %(B^i)^-(2t + 1)
                powB = mod((roots(r)*mod(-(twoT + 1),n)),n);
                %rj(B^i)
                val1 = gfpolyval(rj ,roots(r) ,n ,field);
                %gj'(B^i)
                val2 = gfpolyval(gjDiff ,roots(r) ,n ,field);
                %rj(B^i)/gj'(B^i)
                division = gfdeconv(val1, val2, field);
                %(B^i)^-(2t + 1) * (rj(B^i)/gj'(B^i))
                %v = gfconv(powB,division,field);
                e(roots(r)+1) = gfconv(powB,division,field);
            end

            %calculate the decoded word c = r+e
            decoded = gfadd(recWord,e, field);

        end
    end

    %check if its a codeword
    [quot,remd] = gfdeconv(decoded,g,field);
    if~(remd == -Inf)
        disp('Decoded word is not a valid codeword!');
    end
end
```

gfeuclidRS.m

```matlab
function [g,r] = gfeuclidRS( a, b, t, field)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Performs the Euclid Algorithm on two polynomials a and b            %
% in the field until the degree of the remainder                      %
% polynomial is less than t                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Input:                                                               %
%       a, b:   matrix representing polynomial format in field       %
%               i.e. [1 0 2 3] for A + x + A^2x^2 + A^3x^3           %
%               where A is primitive element in the field            %
%       t:      the degree of the remainder should be <t             %
%       field:  list of all elements in the field p^m                %
%               generated using gftuple                              %
%Output:                                                              %
%       g, r:   matrix representing polynomial format in field       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %initialize the values of r0,f0,g0,r1,f1,g1
    r0 = a;
    f0 = 0;      %represents 1;
    g0 = -Inf;  %represents 0;

    r1 = b;
    f1 = -Inf;   %represents 0;
    g1 = 0;      %represents 1;

    %divide r1 by r0
    [q2, r2] = gfdeconv(r0,r1,field);
    %update values of f2,g2
    f2 = gfsub(f0, gfconv(q2,f1,field), field);
    g2 = gfsub(g0, gfconv(q2,g1,field), field);
    %rearrange the values
    r0 = r1;
    r1 = r2;
    g0 = g1;
    g1 = g2;
    f0 = f1;
    f1 = f2;

    %keep on dividing, updating and rearranging until deg(r2)< t
    while  not( length(r1) < t+1 )%|| r1(t+1)== -Inf)
        [q2, r2] = gfdeconv(r0,r1,field);
        f2 = gfsub(f0, gfconv(q2,f1,field), field);
        g2 = gfsub(g0, gfconv(q2,g1,field), field);

        r0 = r1;
        r1 = r2;
        g0 = g1;
        g1 = g2;
        f0 = f1;
        f1 = f2;
    end

    g = g2;
    r = r2;
end
```

gfallroots.m

```
function roots = gfallroots(poly, n, field)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct the list of zeros or roots of a polynomial                 %
% in the field 'field'                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Input:                                                                %
%       poly:   matrix representing polynomial format in field         %
%               i.e. [1 0 2 3] for A + x + A^2x^2 + A^3x^3             %
%               where A is primitive element in the field              %
%       n:      p^m-1 in the field p^m                                 %
%       field:  list of all elements in the field p^m                  %
%               generated using gftuple                                %
%Output:                                                               %
%       roots:  the list of roots of the polynomial                    %
%               representing polynomail format                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    roots = [];
    %for each alpha^num in the field
    for num = 0:n-1
       %evaluate the polynomial
       value = gfpolyval(poly,num,n,field);
       %if evaluated to zero than its a root
       if(value == -Inf)
          roots = [roots num] ;
       end
    end
end
```

gfdifferentiate.m

```
function diff = gfdifferentiate(poly)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Differentiate a polynomial with respect to x                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Input:                                                                %
%       poly:   matrix representing polynomial format in field         %
%               i.e. [1 0 2 3] for A + x + A^2x^2 + A^3x^3             %
%               where A is primitive element in the field              %
%Output:                                                               %
%       diff:   differentiated polynomial with respect to x            %
%               in matrix representing polynomial format in field      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%calcualte the length of the polynomial
len = length(poly);

for pow = 1:len-1
        %all the even powers are zero
        if mod(pow,2) == 0
            diff(pow) = -Inf;
        %coefficient of x^i will be the
```

```
        %coefficient of x^i+1
        else
            diff(pow) = poly(pow+1);
        end
end
```

gfpolyval.m

```
function val = gfpolyval(poly,point,n,field)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluate a polynomial at given point in the field 'field'          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Input:                                                               %
%       poly:   matrix representing polynomial format in field       %
%               i.e. [1 0 2 3] for A + x + A^2x^2 + A^3x^3           %
%               where A is primitive element in the field            %
%       point:  i means alpha^i                                      %
%       n:      p^m-1 in the field p^m                               %
%       field:  list of all elements in the field p^m                %
%               generated using gftuple                              %
%Output:                                                              %
%       val:    the result of evaluation as the power of alpha       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    len = length(poly);
    val = -Inf;
    for position = 0:len-1
        %(alpha^point)^postion
        xPow = mod(point*position,n);
        %r(position+1).(alpha^point)^postion
        xPos = gfmul(poly(position+1),xPow,field);
        %summation of all products
        val = gfadd(val,xPos,field);
    end
end
```

evaluatePerformance.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This file checks the performance of the (255,239) Reed-Solomon decoder %
% It takes 100 frames of zero codewords                                   %
% Randomly creates same number of errors in each frame                    %
% Then decodes each frame and checks if it is decoded successfully        %
% or cannot be decoded or decoded into wrong codeword                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
% taking the parameters
n = 255;
k = 239;
errorNum = 9;
```

```
%generate a list of elements of GF(2^m)
field = gftuple([-1:2^8-2]',8,2);

%generate the generator polynomial
g = generatorPolynomial(8, field);

%a zero codeword
allEmpty(1:n) = -Inf;

%generating random errors in each frame of length n
recFrame = randerr(100,n,errorNum);

%for each frame
for(frame = 1:100)

    disp(' ');
    disp(sprintf('Processing frame.. %d',frame));

    %change the format to field format
    for (i = 1:n)
        if (recFrame(frame,i)== 0)
            recFrame(frame,i) = -Inf;
        else
            recFrame(frame,i) = 0;
        end
    end

    disp('Received word:')
    for bit = 0:16

disp(sprintf('%s',num2str(recFrame(frame,1+(bit*15):15+(bit*15)))));
    end

    %disp('Sending the code');
    send = recFrame(frame,:);
    %decode the word with errors
    DECODED = RSdecoder(send);

    disp('Decoded word:')
    for bit = 0:16
        disp(sprintf('%s',num2str(DECODED(1+(bit*15):15+(bit*15)))));
    end

    %if it is decoded to the zero word
    if (isequal(DECODED,allEmpty))
        disp('      Succesful Decoding')
    %if it cannot be decoded and returned as it is
    elseif (isequal(DECODED,send))
        disp('      No Change')
    %if it is decoded into another codeword
    else
        disp('      Decoding Error')
    end

end
```

testDecoder.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This file checks the performance of the (255,239) Reed-Solomon decoder%
% It takes one frame of randomly generated codeword                     %
% Randomly creates some errors in the codeword                          %
% Then uses the decoder to decode the code and                          %
% checks if it is decoded successfully                                  %
% or cannot be decoded or decoded into wrong codeword                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;

%generate a list of elements of GF(2^m)
field = gftuple([-1:2^8-2]',8,2);

%generate the generator polynomial
g = generatorPolynomial(8, field);

%generate random data
info = randint(1,239,[-1 255-1]);

for i = 1:239
    if (info(i) < 0)
        info(i) = -Inf;
    end
end

%encoding information bits
encoded = RSencoder(info);

disp('Sending the code');
send = encoded;

field = gftuple([-1:2^8-2]',8,2);

%creating random errors
 send(3) = gfadd(encoded(3),randint(1,1,[-1 255-1]),field);
 send(5) = gfadd(encoded(3),randint(1,1,[-1 255-1]),field);
 send(15) = gfadd(encoded(15),randint(1,1,[-1 255-1]),field);
 send(67) = gfadd(encoded(3),randint(1,1,[-1 255-1]),field);
 send(122) = gfadd(encoded(122),randint(1,1,[-1 255-1]),field);
 send(141) = gfadd(encoded(141),randint(1,1,[-1 255-1]),field);
 send(167) = gfadd(encoded(167),randint(1,1,[-1 255-1]),field);
 send(188) = gfadd(encoded(188),randint(1,1,[-1 255-1]),field);
 send(207) = gfadd(encoded(207),randint(1,1,[-1 255-1]),field);
 send(247) = gfadd(encoded(247),randint(1,1,[-1 255-1]),field);

%uses decoder to decode
DECODED = RSdecoder(send);

%checks the result of decoding
if (isequal(DECODED,encoded))
    disp('Succesful Decoding')
elseif (isequal(DECODED,send))
    disp('No Change')
else
    disp('Decoding Error')
end
```