ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά 3ης Εργαστηριακής Άσκησης στις Ψηφιακές Επικοινωνίες ΙΙ

«Κωδικοποίηση Reed-Solomon»

Μπουφίδης Ιωάννης 03120162

Ερώτημα 1

Έστω κώδικας R-S (n,k) με $n = 31$ και δυνατότητα διόρθωσης $t = 3$ λαθών. Μπορούμε να υπολογίσουμε τα εξής:

- $n = 2^m - 1 = 31 \rightarrow m = 5$
- $k = n - 2t = 31 - 6 = 25$

Επομένως, οι πράξεις θα γίνονται στο σώμα GF(2^5) και το γενετήριο πολυώνυμο $g(x)$ του κώδικα σχηματίζεται ώστε να έχει ρίζες $2t$ διαδοχικές δυνάμεις του a (όπου a: το πρώτο μη μηδενικό, διάφορο του 1 στοιχείο του πεδίου)

Άρα, $\boldsymbol{g(x) = \prod_{i=1}^{2t=6}(x + a^i) = x^6 + a^{10}x^5 + a^9x^4 + a^{24}x^3 + a^{16}x^2 + a^{24}x + a^{21}}$

Στην συνέχεια, κάνουμε αποκωδικοποίηση μηνύματος μίας λέξης 31 μηδενικών (αφού προσθέσουμε τρία λάθη της επιλογής μας, διάνυσμα $\boldsymbol{e(x)}$).

Εφόσον, $u(x) = 0$ ή $u = (0\ 0\ 0\ \ldots 0\ 0\ 0)$, $r(x) = e(x)$. Έστω, $r(x) = e(x) = a^{19}x^{20} + a^2x^5 + a^{20}x^2$. Η αποκωδικοποίηση πρέπει να γίνει:

(i) Χειρωνακτικά, με βάση τον αλγόριθμο υπολογισμού ΜΚΔ του Ευκλείδη.

Αρχικά, υπολογίζουμε τα $2t = 6$ σύνδρομα ως εξής:

- $S_1 = r(a) = a^{19}a^{20} + a^2a^5 + a^{20}a^2 = a^8 + a^7 + a^{22} = a^{20}$
- $S_2 = r(a^2) = a^{19}a^{\cancel{40}\,9} + a^2a^{10} + a^{20}a^4 = a^{28} + a^{12} + a^{24} = a^{19}$
- $S_3 = r(a^3) = a^{19}a^{\cancel{60}\,29} + a^2a^{15} + a^{20}a^6 = a^{17} + a^{17} + a^{26} = a^{26}$
- $S_4 = r(a^4) = a^{19}a^{\cancel{80}\,18} + a^2a^{20} + a^{20}a^8 = a^6 + a^{22} + a^{28} = a^{29}$
- $S_5 = r(a^5) = a^{19}a^{\cancel{100}\,7} + a^2a^{25} + a^{20}a^{10} = a^{26} + a^{27} + a^{30} = a^{12}$
- $S_6 = r(a^6) = a^{19}a^{\cancel{120}\,27} + a^2a^{30} + a^{20}a^{12} = a^{15} + a + a = a^{15}$

Έπειτα, εφαρμόζουμε τον αλγόριθμο ΜΚΔ του Ευκλείδη

Βήμα i	$s_i(x)$	$t_i(x)$	$u_i(x)$	$\varphi_i(x)$
-1	1	0	$x^{2t} = x^6$	
0	0	1	$S(x) = a^{15}x^5 + a^{12}x^4 + a^{29}x^3 + a^{26}x^2 + a^{19}x + a^{20}$	
1	1	$a^{16}x + a^{13}$	$a^2x^4 + a^{14}x^2 + a^{11}x + a^2$	$a^{16}x + a^{13}$
2	$a^{13}x + a^{10}$	$a^{29}x^2 + a^{12}$	$ax^3 + a^{26}x^2 + x + a$	$a^{13}x + a^{10}$
3	$a^{14}x^2 + a^6x + a^5$	$a^{30}x^3 + a^{24}x^2 + a^{11}x + a^3$	$a^{11}x^2 + a^7x + a^{23}$	$ax + a^{26}$

Παρατηρούμε ότι ο βαθμός του $u_3(x) = 2 < 3 = t$.

Επομένως, $\sigma(x) = \lambda t_3(x) = \lambda(a^{30}x^3 + a^{24}x^2 + a^{11}x + a^3)$ και $\omega(x) = \lambda u_3(x) = \lambda(a^{11}x^2 + a^7x + a^{23})$ με λ τέτοιο ώστε $\sigma_0 = 1$. Άρα, $\lambda = a^{28}$, το οποίο δίνει:

- $\sigma(x) = a^{27}x^3 + a^{21}x^2 + a^8x + 1$
- $\omega(x) = a^8x^2 + a^4x + a^{20}$

Υπολογίζουμε τις ρίζες του $\sigma(x)$:

- $\sigma(1) = a^{15} \neq 0$
- $\sigma(\alpha) = a^{19} \neq 0$

  ...
- $\sigma(a^{11}) = 0$
- ...
- $\sigma(a^{26}) = 0$

  ...
- $\sigma(a^{29}) = 0$

  ...
- $\sigma(a^{31}) = a^{15} \neq 0$

Τα αντίστροφα των ριζών $\beta_1 = (\alpha^{11})^{-1} = \alpha^{20}$, $\beta_2 = (\alpha^{26})^{-1} = \alpha^5$ και $\beta_3 = (\alpha^{29})^{-1} = \alpha^2$ υποδεικνύουν ότι έχουμε σφάλματα στους συντελεστές x^{20}, x^5 και x^2 του κωδικού πολυωνύμου αντίστοιχα.

Υπολογίζουμε την 1η παράγωγο του $\sigma(x)$ ως εξής:

$$\sigma'(x) = \beta_1(1 + \beta_2 x)(1 + \beta_3 x) + \beta_2(1 + \beta_1 x)(1 + \beta_3 x) + \beta_3(1 + \beta_1 x)(1 + \beta_2 x)$$

$$\sigma'(x) = \alpha^{27}x^2 + \alpha^8$$

Οπότε, :

- $e_{20} = \frac{\omega(\beta_1{}^{-1})}{\sigma'(\beta_1{}^{-1})} = \frac{1}{a^{12}} = a^{19}$
- $e_5 = \frac{\omega(\beta_2{}^{-1})}{\sigma'(\beta_2{}^{-1})} = \frac{a^{26}}{a^{24}} = a^2$
- $e_2 = \frac{\omega(\beta_3{}^{-1})}{\sigma'(\beta_3{}^{-1})} = \frac{a^{21}}{a} = a^{20}$

τα οποία είναι πράγματι τα σφάλματα που είχαμε εισαγάγει στις θέσεις μονωνύμων x^{20}, x^5 και x^2 αντίστοιχα.

(ii) με την βοήθεια του MATLAB και την τεχνική της αυτοπαλινδρόμησης. Κάτι τέτοιο μπορούμε να καταφέρουμε μέσω του παρακάτω κώδικα, ο οποίος υλοποιεί και την συστηματική κωδικοποίηση R-S ενός άλλου μηνύματος

```
% EXERCISE on RS decoding
%% ------------Parameters------------- %%
clear all; close all;
n = 31;             % length of codeword
t = 3;              % correctable errors
m = log2(n+1);      % Galois field's degree
k = n-2*t;          % length of data-word
d = primpoly(m);    % primitive polynomial
a = gf(2,m,d);      % the first non-zero, non-one element of the field

%% -------Generator Polynomial-------- %%
g = [1 a];
for i=2:2*t
```

```
    g = conv(g,[1 a^i]);
end

%% ----------RS(n,k) Encoder---------- %%
message_bits = randi([0 1], 1, k*m); % random binary data stream, m(x)
% Combining the bits in fives (elements of the GF(2^5)), we get:
reshaped_m=reshape(message_bits,m,length(message_bits)/m);
message=gf(bi2de(reshaped_m','left-msb'),m)';

x_2t = gf([zeros(1,2*t) 1], m);
shifted_m = conv(x_2t, message); % x^(n-k)*m(x)
[q,b] = deconv(flip(shifted_m), g); % x^(n-k)*m(x)= q(x)*g(x) + b(x)
transmitted = shifted_m + flip(b); % u(x) = x^(n-k)*m(x) + b(x)

disp('Reed-Solomon Encoding');
disp('Data:');
disp(message.x);
disp('Transmitted codeword:');
disp(transmitted.x);
disp('');

%% ---------Noise Application--------- %%
% Decoding will be simulated with an all-zero codeword
received_bits = zeros(1,m*n);
e_pos=[12,13,27,101,102]; % bit-error positions within the
                          % (binary) codeword, from lsb
% e_pos=[12,13,14,25,26,52,54]; % try another error pattern

e_pos=length(received_bits)-e_pos; % bit-error positions, counted from msb
for i=1:length(e_pos)
    received_bits(e_pos(i))=~received_bits(e_pos(i));
end

reshaped_r=reshape(received_bits,m,length(received_bits)/m);
received=gf(bi2de(reshaped_r','left-msb'),m)';


%% -------Syndrome Calculation-------- %%
s = gf([],m,d);
for i=1:2*t
    s(i)=received(n);
    for j=1:n-1
        s(i)=s(i)+(a^i)^j*received(n-j);
    end
end
S=gf([],m,d); cons_s=gf([],m,d);
for i=1:t
    for j=0:t-1
        S=[S s(i+j)];
    end
    cons_s(i)=-s(t+i);
end
S=reshape(S,t,length(S)/t)';

%% ---------Error Calculation--------- %%
if det(S)~=0
    sigma=S^-1*cons_s';
    % Find roots of sigma
    s_roots_exp=[]; % vector of sigma roots exponents (of a)
    for i=1:n-1
        ai=a^i; sr=gf(1,m,d);
        for j=1:t
            sr=sr+sigma(j)*ai^(t-j+1);
        end
        if (sr==0)
```

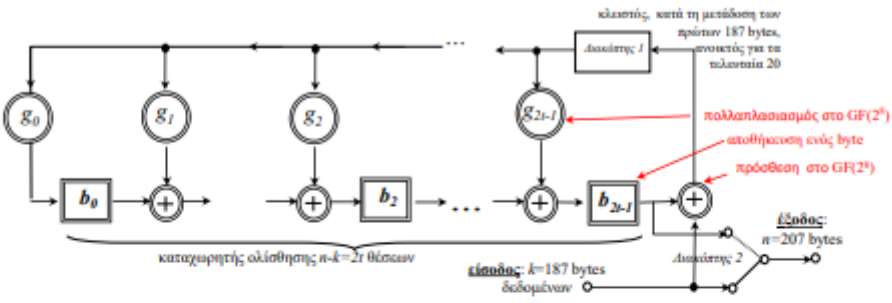
```
            s_roots_exp=[s_roots_exp i];
        end
    end
    b=(a.^s_roots_exp).^-1; % iverses of sigma roots
    er_pos=n-log(b); % error positions from ms digit
    B=gf([],m,d); cons_b=gf([],m,d);
    for i=1:t
        for j=1:t
            B=[B b(j)^i];
        end
        cons_b(i)=s(i);
    end
    B=reshape(B,t,length(B)/t)';
    er=(B^-1*cons_b')';
    corrected=received;
    for i=1:length(er)
        corrected(er_pos(i))=received(er_pos(i))+er(i);
    end

    disp('Reed-Solomon Decoding (all-zero codeword)');
    disp('Received codeword:');
    disp(received.x);
    disp('Corrected codeword:');
    disp(corrected.x);
else
    disp('Determinant equal zero');
end
```

Αν τρέξουμε το πρόγραμμα, θα τυπωθεί σε μορφή μηνύματος τόσο το λανθασμένο ληφθέν διάνυσμα, όσο και το διορθωμένο μήνυμα, όπως φαίνεται παρακάτω

```
Reed-Solomon Decoding (all-zero codeword)
Received codeword:
  Columns 1 through 22

    0    0    0    0    0    0    0    0    0    0    6    0    0    0    0    0    0    0    0    0    0    0

  Columns 23 through 31

    0    0    0    4    0    0   12    0    0

Corrected codeword:
  Columns 1 through 28

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

  Columns 29 through 31

   0   0   0
```

Ερώτημα 2

Στο παρακάτω πλαίσιο, δείχνεται η δομή, καθώς και οι παράμετροι, του κωδικοποιητή R-S στο σύστημα εκπομπής ψηφιακής τηλεόρασης ATSC.



Τα πρώτα k (=187) bytes (δεδομένων) εξέρχονται με το *Διακόπτη 1* κλειστό και το *Διακόπτη 2* στην κάτω θέση. Τα επόμενα 20 bytes (ισοτιμίας) εξέρχονται από τον καταχωρητή ολίσθησης, με το *Διακόπτη 1* ανοικτό. Όλες οι πράξεις γίνονται στο σώμα GF(2^8), με πρωταρχικό πολυώνυμο το $1+x^2+x^3+x^4+x^8$ και πρωταρχικό στοιχείο α. Το γενετήριο πολυώνυμο του κώδικα σχηματίζεται ώστε να έχει ρίζες $2t$ διαδοχικές δυνάμεις του α). Σύμφωνα με το πρότυπο [ATSC2007]-παρ. 6.4.1.2, αυτό είναι*:

$$\prod_{i=0}^{2t-1}(x+\alpha^i)=\alpha^{190}+x^1\alpha^{188}+x^2\alpha^{212}+x^3\alpha^{212}+x^4\alpha^{164}+x^5\alpha^{156}+x^6\alpha^{239}+x^7\alpha^{83}+x^8\alpha^{225}+x^9\alpha^{221}+$$
$$+x^{10}\alpha^{180}+x^{11}\alpha^{202}+x^{12}\alpha^{187}+x^{13}\alpha^{26}+x^{14}\alpha^{163}+x^{15}\alpha^{61}+x^{16}\alpha^{50}+x^{17}\alpha^{79}+x^{18}\alpha^{60}+x^{19}\alpha^{17}+x^{20}$$

Από το παραπάνω ανάπτυγμα, παίρνουμε τους συντελεστές g_k όπως στον παρακάτω πίνακα.

g_0	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_{11}	g_{12}	g_{13}	g_{14}	g_{15}	g_{16}	g_{17}	g_{18}	g_{19}
α^{190}	α^{188}	α^{212}	α^{212}	α^{164}	α^{156}	α^{239}	α^{83}	α^{225}	α^{221}	α^{180}	α^{202}	α^{187}	α^{26}	α^{163}	α^{61}	α^{50}	α^{79}	α^{60}	α^{17}
174	165	121	121	198	228	22	187	36	69	150	112	220	6	99	111	5	240	185	152

* Το πολυώνυμο αυτό δεν είναι μοναδικό. Το πρότυπο ξεκινάει από τη μηδενική δύναμη του στοιχείου α. Αν ξεκινήσουμε π.χ. από την πρώτη δύναμη θα πάρουμε άλλους συντελεστές g_k. (να επιβεβαιωθεί)

Κωδικοποιητής R-S στο σύστημα εκπομπής ATSC

Ο κώδικας MATLAB που υπολογίζει τόσο τους συντελεστές του γενετηρίου πολυωνύμου του συγκεκριμένου κωδικοποιητή, όσο και ενός άλλου πολυωνύμου με ρίζες τις δυνάμεις $a^i, i = 1, \dots, 20$ είναι ο εξής:

```
%% -----------------Parameters----------------- %%
clear all; close all;
k = 187;         % length of data-word
n = 20 + 187;    % length of codeword
m = 8;           % Galois field's degree
t = (n-k)/2;     % correctable errors
d=primpoly(m);   % primitive polynomial
a = gf(2,m);     % primitive element

% Generator polynomials are formed in order to
% have 2t consecutive powers of a as roots
%% -------Generator Polynomial (template)------- %%
% g(x)=(1+x)(a+x)(a^2+x)...(a^19+x)
g0 = [a^0 1];
for i=1:2*t-1
    g0 = conv(g0,[a^i 1]);
end
disp("Generator polynomial (template):");
disp(g0.x);


%% ----Generator Polynomial (first root:a^1)---- %%
% g(x)=(a+x)(a^2+x)(a^3+x)...(a^20+x)
g1 = [a 1];
for i=2:2*t
    g1 = conv(g1,[a^i 1]);
end
```

```
disp("Generator polynomial (first root: a^1):");
disp(g1.x);

%% ---------Different Polynomials Check--------- %%
if ~isequal(g0, g1)
    disp("Generator polynomials are not equal");
else
    disp("Generator polynomials are equal");
end
```

Στην συνέχεια, με βάση τα παρακάτω μηνύματα που τυπώνονται μέσω του κώδικα, μπορούμε να επιβεβαιώσουμε τον σχετικό πίνακα του προτύπου (πλαίσιο παραπάνω), αλλά και να επαληθεύσουμε ότι το δεύτερο πολυώνυμο είναι διαφορετικό από το πρώτο

```
Primitive polynomial(s) =

D^8+D^4+D^3+D^2+1
Generator polynomial (template):
  Columns 1 through 18

   174   165   121   121   198   228    22   187    36    69   150   112   220     6    99   111     5   240

  Columns 19 through 21

   185   152     1

Generator polynomial (first root: a^1):
  Columns 1 through 18

    89   166   244   122   150   179    71   217   139   247   174   178   100    39   229    97    80   211

  Columns 19 through 21

   222    45     1

Generator polynomials are not equal
```

Ερώτημα 3

Έστω κώδικας R-S (207,187) με και δυνατότητα διόρθωσης $\frac{n-k}{2} = 10$ λαθών.

Οι πράξεις θα γίνονται στο σώμα GF(2^8) και το γενετήριο πολυώνυμο $g(x)$ του κώδικα σχηματίζεται ώστε να έχει ρίζες $2t$ διαδοχικές δυνάμεις του a (όπου a: το πρώτο μη μηδενικό, διάφορο του 1 στοιχείο του πεδίου)

Άρα, $\boldsymbol{g(x) = \prod_{i=0}^{2t-1}(x + a^i)}$, με αποτέλεσμα να προκύπτει το $g(x)$ του πρότυπου ATSC του **Ερωτήματος 2**.

Στην συνέχεια, κάνουμε αποκωδικοποίηση μηνύματος μίας λέξης 207 μηδενικών (αφού προσθέσουμε τρία λάθη της επιλογής μας, διάνυσμα $\boldsymbol{e(x)}$).

Εφόσον, $u(x) = 0$ ή $u = (0\ 0\ 0 \ldots 0\ 0\ 0)$, $r(x) = e(x)$. Έστω, $r(x) = e(x) = \alpha x^2 + \alpha x + \alpha$. Η αποκωδικοποίηση πρέπει να γίνει:

(i) Χειρωνακτικά, με βάση τον αλγόριθμο υπολογισμού ΜΚΔ του Ευκλείδη.

Αρχικά, υπολογίζουμε τα $2t = 20$ σύνδρομα ως εξής:

- $S_1 = r(a) = a^{199}$

- $S_2 = r(a^2) = a^{142}$
- $S_3 = r(a^3) = a^{153}$
- $S_4 = r(a^4) = a^{28}$
- $S_5 = r(a^5) = a^{151}$
- $S_6 = r(a^6) = a^{50}$
- $S_7 = r(a^7) = a^{154}$
- $S_8 = r(a^8) = a^{55}$
- $S_9 = r(a^9) = a^{240}$
- $S_{10} = r(a^{10}) = a^{46}$
- $S_{11} = r(a^{11}) = a^{26}$
- $S_{12} = r(a^{12}) = a^{99}$
- $S_{13} = r(a^{13}) = a^{8}$
- $S_{14} = r(a^{14}) = a^{52}$
- $S_{15} = r(a^{15}) = a^{254}$
- $S_{16} = r(a^{16}) = a^{109}$
- $S_{17} = r(a^{17}) = a^{171}$
- $S_{18} = r(a^{18}) = a^{224}$
- $S_{19} = r(a^{19}) = a^{247}$
- $S_{20} = r(a^{20}) = a^{91}$

Έπειτα, εφαρμόζουμε τον αλγόριθμο ΜΚΔ του Ευκλείδη

Βήμα i	$s_i(x)$	$t_i(x)$	$u_i(x)$	$\varphi_i(x)$
-1	1	0	$x^{2t} = x^{20}$	
0	0	1	$S(x)$	
1	1	$a^{164}x + a^{65}$	$x^{20} mod\ S(x)$	$a^{164}x + a^{65}$
2	$a^{176}x + a^{81}$	$a^{85}x^2 + a^{86}x + a^{217}$	$S(x)\ mod\ u_1(x)$	$a^{176}x + a^{81}$
3	$a^{105}x^2 + a^{90}x + a^{57}$	$a^{14}x^3 + a^{210}x^2 + a^{209}x + a^{11}$	$a^{15}x^2 + a^{210}$	$a^{184}x + a^2$

Παρατηρούμε ότι ο βαθμός του $u_3(x) = 2 < 10 = t$.

Επομένως, $\sigma(x) = \lambda t_3(x) = \lambda(a^{14}x^3 + a^{210}x^2 + a^{209}x + a^{11})$ και $\omega(x) = \lambda u_3(x) = \lambda(a^{15}x^2 + a^{210})$ με λ τέτοιο ώστε $\sigma_0 = 1$. Άρα, $\lambda = a^{244}$, το οποίο δίνει:

- $\sigma(x) = a^3x^3 + a^{199}x^2 + a^{198}x + 1$
- $\omega(x) = a^4x^2 + a^{199}$

Εφόσον έχουμε 207 bits και όχι 255, δηλαδή το κωδικό πολυώνυμο (άρα και το πολυώνυμο λάθους) μπορεί να είναι το πολύ 206ου βάθμου, θα ψάξουμε για ρίζες a^i του $\sigma(x)$ στο εύρος εκθετών $i \in [49, 255]$.

Αυτό προκύπτει απ΄το γεγονός, ότι σε περίπτωση που προκύψει κάποια ρίζα a^i του $\sigma(x)$ με $i < 49$, θα έχει εντοπιστεί σφάλμα στον συντελεστή x^{255-i} του κωδικού πολυωνύμου. Όμως, για $i < 49$, $255 - i > 206$, πράγμα εν προκειμένω αδύνατον.

Οπότε, υπολογίζουμε τις ρίζες του $\sigma(x)$:

- $\sigma(a^{49}) = a^{182} \neq 0$
- $\sigma(a^{50}) = a^{126} \neq 0$

  ...
- $\sigma(a^{253}) = 0$
- $\sigma(a^{254}) = 0$
- $\sigma(a^{255}) = 0$

Τα αντίστροφα των ριζών $\beta_1 = (\alpha^{253})^{-1} = \alpha^2$, $\beta_2 = (\alpha^{254})^{-1} = \alpha^1$ και $\beta_3 = (\alpha^{255})^{-1} = \alpha^0$ υποδεικνύουν ότι έχουμε σφάλματα στους συντελεστές x^2, x^1 και x^0 του κωδικού πολυωνύμου αντίστοιχα.

Για το συγκεκριμένο πολυώνυμο υπολογισμού σφαλμάτων ισχύει ότι:

$$\sigma(x) = (1 + \beta_1 x)(1 + \beta_2 x)(1 + \beta_3 x)$$

Οπότε, υπολογίζουμε την 1η παράγωγο του $\sigma(x)$ ως εξής:

$$\sigma'(x) = \beta_1(1 + \beta_2 x)(1 + \beta_3 x) + \beta_2(1 + \beta_1 x)(1 + \beta_3 x) + \beta_3(1 + \beta_1 x)(1 + \beta_2 x)$$

$$\sigma'(x) = \alpha^3 x^2 + \alpha^{198}$$

Οπότε, :

- $e_2 = \frac{\omega({\beta_1}^{-1})}{\sigma'({\beta_1}^{-1})} = \frac{a^{75}}{a^{74}} = a$
- $e_1 = \frac{\omega({\beta_2}^{-1})}{\sigma'({\beta_2}^{-1})} = \frac{a^{51}}{a^{50}} = a$
- $e_0 = \frac{\omega({\beta_3}^{-1})}{\sigma'({\beta_3}^{-1})} = \frac{a^{76}}{a^{75}} = a$

τα οποία είναι πράγματι τα σφάλματα που είχαμε εισαγάγει στις θέσεις μονωνύμων x^2, x^1 και x^0 αντίστοιχα.

(ii) με την βοήθεια του MATLAB και την τεχνική της αυτοπαλινδρόμησης. Κάτι τέτοιο μπορούμε να καταφέρουμε μέσω του παρακάτω κώδικα, ο οποίος υλοποιεί και την συστηματική κωδικοποίηση R-S ενός άλλου μηνύματος

```
% EXERCISE on RS decoding
%% ------------Parameters------------- %%
clear all; close all;
n = 207;              % length of codeword
t = 10;               % correctable errors
m = 8;                % Galois field's degree
k = 187;              % length of data-word
d = primpoly(m);      % primitive polynomial
a = gf(2,m,d);        % the first non-zero, non-one element of the field

%% -------Generator Polynomial-------- %%
% g(x)=(1+x)(a+x)(a^2+x)...(a^19+x)
g = [a^0 1];
for i=1:2*t-1
    g = conv(g,[a^i 1]);
end

%% ----------RS(n,k) Encoder---------- %%
message_bits = randi([0 1], 1, k*m); % random binary data stream, m(x)
% Combining the bits in fives (elements of the GF(2^5)), we get:
reshaped_m=reshape(message_bits,m,length(message_bits)/m);
message=gf(bi2de(reshaped_m','left-msb'),m)';

x_2t = gf([zeros(1,2*t) 1], m);
shifted_m = conv(x_2t, message); % x^(n-k)*m(x)
[q,b] = deconv(flip(shifted_m), g); % x^(n-k)*m(x)= q(x)*g(x) + b(x)
transmitted = shifted_m + flip(b); % u(x) = x^(n-k)*m(x) + b(x)

disp('Reed-Solomon Encoding');
disp('Data:');
disp(message.x(171:187));
disp('Transmitted codeword:');
disp(transmitted.x(191:207));
```

```
disp('');

%% ---------Noise Application--------- %%
% Decoding will be simulated with an all-zero codeword
received_bits = zeros(1,m*n);
e_pos=[1,9,17]; % bit-error positions within the
                            % (binary) codeword, from lsb
% e_pos=[12,13,14,25,26,52,54]; % try another error pattern

e_pos=length(received_bits)-e_pos; % bit-error positions, counted from msb
for i=1:length(e_pos)
    received_bits(e_pos(i))=~received_bits(e_pos(i));
end

reshaped_r=reshape(received_bits,m,length(received_bits)/m);
received=gf(bi2de(reshaped_r','left-msb'),m)';


%% -------Syndrome Calculation-------- %%
s = gf([],m,d);
for i=1:2*t
    s(i)=received(n);
    for j=1:n-1
        s(i)=s(i)+(a^i)^j*received(n-j);
    end
end
S=gf([],m,d); cons_s=gf([],m,d);
for i=1:t
    for j=0:t-1
        S=[S s(i+j)];
    end
    cons_s(i)=-s(t+i);
end
S=reshape(S,t,length(S)/t)';

%% ---------Error Calculation--------- %%
if det(S)~=0
    sigma=S^-1*cons_s';
    % Find roots of sigma
    s_roots_exp=[]; % vector of sigma roots exponents (of a)
    for i=1:n-1
        ai=a^i; sr=gf(1,m,d);
        for j=1:t
            sr=sr+sigma(j)*ai^(t-j+1);
        end
        if (sr==0)
            s_roots_exp=[s_roots_exp i];
        end
    end
    b=(a.^s_roots_exp).^-1; % iverses of sigma roots
    er_pos=n-log(b); % error positions from ms digit
    B=gf([],m,d); cons_b=gf([],m,d);
    for i=1:t
        for j=1:t
            B=[B b(j)^i];
        end
        cons_b(i)=s(i);
    end
    B=reshape(B,t,length(B)/t)';
    er=(B^-1*cons_b')';
    corrected=received;
    for i=1:length(er)
        corrected(er_pos(i))=received(er_pos(i))+er(i);
    end
```

```
    disp('Reed-Solomon Decoding (all-zero codeword)');
    disp('Received codeword:');
    disp(received.x);
    disp('Corrected codeword:');
    disp(corrected.x);
else
    disp('Determinant equal zero');
end
```