



Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

1^ο Σετ Εργαστηριακών Ασκήσεων

Εκπονήθηκε από τον:

Καπογιάννης Κωσταντίνος Α.Μ. 1072521

Πάτρα 2023-24

Μέρος Α΄

1α. Τα διαφορετικά σύμβολα της πηγής είναι εύκολο να βρεθούν, χρησιμοποιώντας την συνάρτηση *unique(I)*, η οποία βρίσκει όλες τις διαφορετικές τιμές της greyscale εικόνας μας. Οι τιμές των συμβόλων φαίνονται παρακάτω:

	1
1	0
2	17
3	34
4	51
5	68
6	85
7	102
8	119
9	136
10	153
11	170
12	187
13	204
14	221
15	238
16	255

Έπειτα, για να βρούμε την πιθανότητα εμφάνισης του κάθε συμβόλου, αρκεί να μετρήσουμε το πλήθος των εμφανίσεων του κάθε συμβόλου και έπειτα να το διαιρέσουμε με το πλήθος των pixels της εικόνας μας. Παρακάτω φαίνεται η πιθανότητα εμφάνισης του κάθε συμβόλου του παραπάνω πίνακα:

	1
1	0.0962
2	0.0814
3	0.0683
4	0.0625
5	0.0768
6	0.0905
7	0.1132
8	0.0906
9	0.0965
10	0.0671
11	0.0389
12	0.0329
13	0.0337
14	0.0259
15	0.0224
16	0.0031

1b. i) Στην συνέχεια, υπολογίζουμε την κωδικοποίηση Huffman των συμβόλων της πηγής, χρησιμοποιώντας την εντολή:

```
huffdict1 = huffmandict(symbols, probabilities);
```

,όπου το **huffdict1** είναι το αλφάβητο της κωδικοποίησης που προέκυψε και τα **symbols, probabilities** είναι τα διανύσματα των συμβόλων και των πιθανοτήτων εμφάνισης αντίστοιχα. Τα σύμβολα μαζί με την κωδικοποίησή τους κατά Huffman, φαίνονται παρακάτω:

	1	2
1	0	[1,1,0]
2	17	[0,0,1,0]
3	34	[0,1,0,0]
4	51	[0,1,1,1]
5	68	[0,0,1,1]
6	85	[0,0,0,0]
7	102	[1,0,0]
8	119	[1,1,1]
9	136	[1,0,1]
10	153	[0,1,0,1]
11	170	[0,0,0,1,1]
12	187	[0,1,1,0,1]
13	204	[0,1,1,0,0]
14	221	[0,0,0,1,0,0]
15	238	[0,0,0,1,0,1,0]
16	255	[0,0,0,1,0,1,1]

Όπως διακρίνουμε και γνωρίζουμε από την θεωρία, ο κώδικας Huffman είναι ένας προθεματικός κώδικας, δηλαδή καμία λέξη δεν είναι πρόθεμα κάποιας άλλης. Η κωδικοποίηση Huffman, καθώς και άλλοι προθεματικοί κώδικες, έχουν το χαρακτηριστικό ότι πετυχαίνουν ρυθμό κωδικοποίησης κοντά στην εντροπία, δηλαδή στο όριο συμπίεσης της της πηγής δίχως απώλειες, κάτι το οποίο περιγράφεται στο θεώρημα του Shannon περί κωδικοποίησης πηγής.

Η εντροπία της κωδικοποίησης της πηγής υπολογίζεται ως εξής:

$$H(\Phi) = - \sum_{i=1}^N p_i \log_2 p_i$$

Στην Matlab, υπολογίζεται με την βοήθεια της συνάρτησης **entropy()**, χρησιμοποιώντας την εντολή:

ent1 = entropy(probabilities);

,όπου probabilities είναι οι πιθανότητες εμφάνισης του κάθε συμβόλου. Η εντροπία της πηγής έχει την τιμή 3.6250

ii) Το μέσο μήκος κώδικα υπολογίζεται από τον τύπο:

$$\bar{L} = \sum_{i=1}^N p(s_i) l(s_i)$$

Υλοποιώντας την συγκεκριμένη συνάρτηση στην Matlab, το μέσο μήκος κώδικα της κατά Huffman κωδικοποίησης των συμβόλων, βρέθηκε ότι είναι **3.8374**. Και εδώ βλέπουμε πόσο αποδοτική είναι η κωδικοποίηση Huffman, αφού προσεγγίζει αρκετά την εντροπία της συγκεκριμένης πηγής, καθώς διαιρώντας την εντροπία με το μέσο μήκος της κωδικοποίησης, προκύπτει ότι η απόδοση είναι η εξής:

$$\eta = \frac{H(\Phi)}{L} = 0.9447$$

, δηλαδή η απόδοση της συγκεκριμένης κωδικοποίησης είναι της τάξεως του 95%, κάτι το οποίο συνοψίζει τα παραπάνω λεγόμενα.

2a. Η εικόνα που έχουμε στην διάθεση μας είναι συγκεκριμένη, με την έννοια ότι οι τιμές που λαμβάνουν τα pixels της εικόνας δεν είναι τυχαία. Συνεπώς, μπορούμε να καταλάβουμε ότι η πιθανότητα εμφάνισης μίας δυάδας συμβόλων, δηλαδή ενός συμβόλου της δεύτερης τάξης επέκτασης της πηγής, δεν θα είναι το γινόμενο των επιμέρους πιθανοτήτων εμφάνισης των δύο αρχικών συμβόλων, καθώς αυτό προϋποθέτει ανεξαρτησία των συμβόλων της αρχικής πηγής. Ένας τρόπος λοιπόν υπολογισμού των πιθανοτήτων εμφάνισης κάθε δυνατής δυάδας συμβόλων, δηλαδή κάθε συμβόλου της δεύτερης τάξης επέκτασης της πηγής, είναι ο εξής:

1. Για κάθε pixel της εικόνας, βλέπουμε τα ζευγάρια που παράγει με τα γειτονικά του pixel (και τα 8, δηλαδή στην ίδια γραμμή στήλη και διαγώνιο), βλέπουμε δηλαδή τις εμφανίσεις των σύνθετων συμβόλων της δεύτερης τάξης επέκτασης της πηγής.
2. Καταγράφουμε αυτές τις εμφανίσεις στο αντίστοιχο σύμβολο.
3. Βρίσκουμε την πιθανότητα εμφάνισης του κάθε σύνθετου διαιρώντας τις εμφανίσεις του συγκεκριμένου συμβόλου με τις συνολικές εμφανίσεις όλων των συμβόλων.

Μια απλοποίηση της παραπάνω μεθόδου θα ήταν να μετράμε μόνο τις εμφανίσεις των ζευγαριών του κάθε pixel, με τον δεξιό του γείτονα, ωστόσο αυτό θα έχει επίπτωση στον προσδιορισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου. Κάνουμε την παραδοχή ότι για κάθε pixel θα παίρνουμε το ζεύγος μόνο από τα δεξιά του και προχωράμε στο μεθεπόμενο. Προκύπτουν έτσι 256 διαφορετικά ζευγάρια τιμών το πολύ.

Το πρώτο βήμα είναι η μετατροπή του μητρώου της εικόνας σε διάνυσμα γραμμή, προκειμένου να απαλείψουμε τις στήλες. Αυτό επιτυγχάνεται με την βοήθεια της εντολής:

```
Ires = reshape(I',[],1);
```

Έπειτα, διαμορφώνουμε αυτό το vector σε μητρώο δύο στηλών, όπου τα στοιχεία της κάθε γραμμής είναι ένα σύνθετο σύμβολο, όπως συναντάται στο διάνυσμα γραμμή:

```
Iproc = [Itest(1:2:end),Itest(2:2:end)];
```

Ένα κομμάτι από το μητρώο **Iproc** φαίνεται παρακάτω:

	1	2
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	17
15	0	0
16	0	0
17	17	17
18	17	17
19	17	17
20	17	17
21	0	0

Βασιζόμενοι στο μητρώο **Iproc**, βρίσκουμε τα μοναδικά ζευγάρια τιμών που βρίσκονται στο **Iproc**, καθώς και κάνουμε ένα mapping του κάθε είδους ζευγαριού με μία συγκεκριμένη τιμή (τιμές του διανύσματος **row_labels**). Έτσι, προκύπτουν όλα τα ζευγάρια συμβόλων, δηλαδή τα σύμβολα της δεύτερης τάξης επέκτασης της πηγής, που παρατηρώντας το μέγεθος του μητρώου **symbols2**, είναι 174 στο πλήθος. Η εντολή φαίνεται παρακάτω:

```
[symbols2,column_labels,row_labels] = unique(Iproc, "rows");
```

και ένα κομμάτι των συμβόλων φαίνεται παρακάτω:

	1	2
1	0	0
2	0	17
3	0	34
4	0	51
5	0	68
6	0	85
7	0	136
8	17	0
9	17	17
10	17	34
11	17	51
12	17	68
13	17	85
14	17	102
15	17	119
16	17	136
17	17	153
18	34	0
19	34	17
20	34	34
21	34	51

Έπειτα, μετράμε όλες τις εμφανίσεις κάθε διαφορετικού συμβόλου μετρώντας πόσες φορές εμφανίζεται η ετικέτα του στο διάνυσμα **row_labels**. Αυτό γίνεται χρησιμοποιώντας την εντολή:

```
counts2 = histcounts(row_labels, max(row_labels));
```

Συνεπώς τώρα ξέροντας τις φορές εμφάνισης του κάθε σύνθετου συμβόλου, μπορούμε να υπολογίσουμε την πιθανότητα εμφάνισης του, διαιρώντας τις με τις συνολικές, δηλαδή:

```
totalCount = sum(counts2);  
probabilities2 = counts2 ./ totalCount;
```

Κάποιες από τις πιθανότητες των συμβόλων φαίνονται παρακάτω:

	1
1	0.0801
2	0.0133
3	0.0019
4	4.6667e-04
5	4.6667e-04
6	2.0000e-04
7	6.6667e-05
8	0.0140
9	0.0467
10	0.0150
11	0.0029
12	0.0012
13	7.3333e-04
14	2.6667e-04
15	2.0000e-04
16	6.6667e-05
17	6.6667e-05
18	0.0013
19	0.0157
20	0.0321
21	0.0121

2b. Και εδώ η εντροπία υπολογίζεται μέσω του διανυσμάτων των πιθανοτήτων που βρήκαμε στο παραπάνω ερώτημα, μόνο που εδώ επιλέχθηκε ο υπολογισμός της εντροπίας να γίνει χωρίς την βοήθεια κάποιας έτοιμης συνάρτησης Matlab. Συνεπώς ο κώδικας για τον υπολογισμό της δίνεται παρακάτω:

```
for i = 1 : 174
    ent2 = ent2-(probabilities2(i,1) * log2(probabilities2(i,1)));
end
```

Έπειτα, για να υπολογιστεί το λεξικό της κωδικοποίησης κατά Huffman, το μητρώο **symbols2**, έπρεπε να υποστεί αλλαγές, προκειμένου να γίνει cell ενός κελιού, όρισμα κατάλληλο για την συνάρτηση **huffmandict** της matlab. Συνεπώς αυτή η μετατροπή φαίνεται παρακάτω:

```
symbols2form = cell(174,1);

for i = 1 : 174
    symbols2form{i} = [num2str(symbols2(i,1)), '-',
        num2str(symbols2(i,2))];
end
```

Έπειτα, με παρόμοιο τρόπο όπως στο παραπάνω ερώτημα, υπολογίζουμε το λεξικό της κωδικοποίησης Huffman της επέκτασης της πηγής, και το μέσο μήκος αυτής:

```
huffdict2 = huffmandict(symbols2form,probabilities2);

lbar2 = 0;
for i = 1:174
    lbar2 = lbar2 + (probabilities2(i,1) *
        length(huffdict2{i,2}));
end
```

2c. Τόσο η εντροπία, όσο και το μέσο μήκος κώδικα της δεύτερης τάξης επέκτασης της πηγής βρέθηκε μεγαλύτερο από τα αντίστοιχα μεγέθη των αρχικών συμβόλων της πηγής. Μάλιστα βλέπουμε ότι η αποδοτικότητα της δεύτερης τάξης επέκτασης της πηγής είναι **0.9956**, δηλαδή της τάξης του **99.5%**. Καταλαβαίνουμε συνεπώς, ότι ενώ το κάθε σύνθετο σύμβολο χρειάζεται περισσότερα bit για την κωδικοποίηση του, μας δίνεται εγγύηση ότι η συγκεκριμένη κωδικοποίηση είναι αρκετά ικανοποιητική.

3a. Ο τύπος δεν ισχύει στην περίπτωση μας διότι οι τιμές των συμβόλων της πηγής δεν είναι ανεξάρτητες μεταξύ τους, δηλαδή δεν υπάρχει στατιστική ανεξαρτησία μεταξύ των συμβόλων της πηγής. Συγκεκριμένα, η τιμή φωτεινότητας ενός pixel, άρα η τιμή του ενός συμβόλου, εξαρτάται άμεσα από τις τιμές των διπλανών του συμβόλων, καθώς η εικόνα δεν είναι τυχαία, αλλά απεικονίζει μια συγκεκριμένη εικόνα. Συνεπώς παρόλο που η εντροπία της δεύτερης τάξης επέκτασης της πηγής αυξάνεται, το ίδιο και η αποδοτικότητα, δεν είναι η διπλάσια της αρχικής πηγής.

3β. Από την θεωρία ξέρουμε ότι για κάθε πηγή και κάθε επέκταση της μπορεί να κατασκευαστεί προθεματικός κώδικας του οποίου το μέσο μήκος φράσσεται σύμφωνα με την παρακάτω σχέση:

$$H(X^n) \leq \bar{L} \leq H(X^n) + 1$$

Γνωρίζοντας το ακόλουθο και σύμφωνα με τα αποτελέσματα που μας έχουν προκύψει, η ανισότητα παίρνει την ακόλουθη μορφή:

Για την εντροπία της πηγής:

$$3.625 \leq \bar{L} \leq 4.625$$

Για την εντροπία της δεύτερης τάξης επέκτασης της πηγής:

$$5.793 \leq \bar{L} \leq 6.793$$

4. Κωδικοποιούμε την εικόνα, το διάνυσμα γραμμή που υλοποιήσαμε, χρησιμοποιώντας την συνάρτηση **huffmanenco** της matlab. Συγκεκριμένα, ο κώδικας που υλοποιήθηκε για το συγκεκριμένο ερώτημα είναι ο εξής:

```
encSrc = huffmanenco(Ires,huffdict1);
decSrc = huffmandeco(encSrc, huffdict1);
bI = reshape((dec2bin(typecast(I(:),'uint8'),4)-'0').',1,[]);
binaryToHuffmanRatio = length(encSrc) / length(bI);
```

Από το κώδικα φαίνεται ότι κωδικοποιούμε την εικόνα χρησιμοποιώντας το λεξικό των συμβόλων και έπειτα την αποκωδικοποιούμε χρησιμοποιώντας το ίδιο λεξικό. Παρατηρούμε ότι το διάνυσμα που παράγεται μετά την αποκωδικοποίηση είναι ίδιο με το αρχικό, διαπιστώνοντας έτσι την εγκυρότητα της κωδικοποίησης Huffman. Έπειτα, συγκρίνουμε τον αριθμό των bits που χρειάστηκε η κωδικοποίηση Huffman σε σχέση με τον αριθμό bits που χρειάζεται η δυαδική αναπαράσταση της. Συγκεκριμένα, με πρόχειρους υπολογισμούς μπορούμε να θεωρήσουμε ότι η κωδικοποίηση Huffman θα μας επιφέρει κωδικοποίηση περίπου 114.900 bits για μια εικόνα 30.000 συμβόλων, αφού το μέσος μήκος κώδικα είναι περίπου 3.84 bits/symbol. Πράγματι, κάνοντας την κωδικοποίηση βρίσκουμε ότι το διάνυσμα περιέχει 115.121 bits, ενώ η δυαδική αναπαράσταση της πηγής, χρησιμοποιώντας την εντολή που μας δόθηκε, μας επιφέρει διάνυσμα των 240.000 bits, συνεπώς ο λόγος που προκύπτει είναι $J = 0.47$, δηλαδή βλέπουμε ότι η κωδικοποίηση Huffman είναι κοντά δύο φορές αποδοτικότερη από την δυαδική αναπαράσταση.

5. Προκειμένου να προσδιορίσουμε τη πιθανότητα μετάβασης του καναλιού περνάμε το διάνυσμα που προέκυψε από την κωδικοποίηση κατά Huffman μέσα από το κανάλι. Αυτό επιτυγχάνεται με την βοήθεια της εντολής:

```
output = binary_symmetric_channel(encSrc);
```

,όπου το encSrc είναι το διάνυσμα που προέκυψε από την κωδικοποίηση Huffman. Συνεπώς τώρα μπορούμε να συγκρίνουμε εύκολα τα δύο διανύσματα, προκειμένου να προσδιορίσουμε πόσα bit παραλήφθηκαν λάθος. Ο προσδιορισμός αυτό γίνεται με την χρήση της συνάρτησης xor, όπου κάνει το xor μεταξύ των στοιχείων των δύο διανυσμάτων και επιστρέφει 1 εάν τα bit είναι διαφορετικά μεταξύ τους. Έπειτα, μετράει τους άσους σε αυτό το διάνυσμα που προκύπτει, προκειμένου να διαιρέσουμε αυτό τον αριθμό με το μέγεθος του διανύσματος και να προσδιορίσουμε την πιθανότητα μετάβασης, την οποία μετράμε ότι είναι $p = 0.11$. Ο κώδικας για το συγκεκριμένο ερώτημα φαίνεται παρακάτω:

```
output = binary_symmetric_channel(encSrc);
```

```
comparison = xor(encSrc, output);  
comparisonCounts = sum(comparison);
```

```
channelProb = comparisonCounts / length(encSrc);
```

, όπου **output** είναι το διάνυσμα εξόδου του δυαδικού καναλιού μας, το **comparison** είναι το διάνυσμα που προκύπτει από την σύγκριση των δύο διανυσμάτων εισόδου και εξόδου, και η **channelProb** είναι η ζητούμενη πιθανότητα. Έπειτα, μπορούμε πολύ εύκολα να βρούμε τη χωρητικότητα του καναλιού, αφού ξέρουμε ότι για ένα δυαδικό συμμετρικό κανάλι δίνεται από την εξίσωση:

$$C(p) = 1 - H_b(p), \quad \text{όπου } H_b(p) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

Έτσι μπορούμε να υπολογίσουμε εύκολα την χωρητικότητα του καναλιού, γνωρίζοντας την πιθανότητα μετάβασης, συνεπώς με τον παρακάτω κώδικα:

```
capacity = 1 + channelProb*log2(channelProb) + (1 -  
channelProb)*log2(1 - channelProb);
```

βρίσκουμε το $\text{capacity} = 0.4732$. Τέλος, χρησιμοποιώντας την συνάρτηση $I(X; Y) = H(Y) - H(Y|X)$, υπολογίζουμε ότι η αμοιβαία πληροφορία μεταξύ της εισόδου και της εξόδου του καναλιού είναι 0.4717, δηλαδή είναι περίπου ίση με την χωρητικότητα του καναλιού, δεδομένου ότι η πιθανότητα λάθους είναι αρκετά μικρή.

Μέρος Β'

1. Οι συναρτήσεις που υλοποιήσαμε για το συγκεκριμένο σύστημα, καθώς και τα σχόλια επεξήγησης τους, φαίνονται παρακάτω:

a) **my_quantizer:**

```
function [y_cap, centers] = my_quantizer(y, N, min_value, max_value)
% MY_QUANTIZER inputs:
% 1. y: the input signal to be quantized.
% 2. N: the number of bits used for the representation of the
%    representative values.
% 3. min_value: maximum value of quantization.
% 4. max_value: minimum value of quantization.
%
% MY_QUANTIZER outputs:
% 1. yn: the quantized signal.
% 2. centers: the representative values of each quantization level.

% We calculate the number of quantization levels (dynamic regions).
L = 2^N;

% We calculate the step size
D = (max_value - min_value) / L;

% We now have L levels of quantization each of size D. We now calculate the
% center of each quantization level as the middle of each level.

centers = min_value + D/2 : D : max_value - D/2;

% We preallocate the space for the quantized signal
y_cap = zeros(size(y));

% Quantize each element of the input signal. This is achieved by
% creating a vector with the absolute difference of the value with each of
% the centers. Then we find the position of the minimum value in that
% vector and we insert the value of the center in the same position in the
% quantized output signal.
for i = 1:length(y)
    [~, idx] = min(abs(centers - y(i)));
    y_cap(i) = centers(idx);
end
end
```

b) **prediction_filter:**

```
function [r, R, a] = prediction_filter(p, x)
% Prediction_filter inputs:
% 1. p: the number of previous values of the signal we want to keep.
```

```

% 2. x: the input signal.
%
% Prediction_filter outputs:
% 1. r: self-correlation vector of size px1.
% 2. R: self-correlation matrix of size pxp.
% 3. a: vector of the prediction filter coefficients.

N = length(x);
frac = 1/(N-p);
r = zeros(p,1);
R = zeros(p,p);
a = zeros(p,1);

% We calculate the rx(i) which go to r(i)
for i = 1 : p
    sum = 0;
    for n = p+1 : N
        sum = sum + (x(n)*x(n-i));
    end
    r(i) = sum*frac;
end

% We calculate the Rx(i - j) which go to the R(i,j)
for i = 1 : p
    for j = 1 : p
        sum = 0;
        for n = p+1 : N
            sum = sum + x(n - j)*x(n - i);
        end
        R(i,j) = sum*frac;
    end
end

% We calculate the coefficients, using the Yule Walker equations.
a = R\r;

end

```

c) predictor:

```

function [predicted_signal] = predictor(a,n,y_cap_tone)
% predictor inputs:
% 1. a: The prediction coefficients used to predict the signal's values.
% 2. n: The size of the signal.
% 3. y_cap_tone: the processed output signal, which is the input of the
% predictor
%
% predictor outputs:
% 1. predicted_signal: the signal, whose values are the predictions of
% the input signal.

%preallocate the space of the vector
predicted_signal = zeros(n,1);

%first value is zero because the first iteration has no previous value of
%the input signal, to calculate a prediction
predicted_signal(1) = 0;

% i is equal to p+1 or length(a)+1, to ensure the i-j parameter inside the

```

```

% loop is positive. This means the first p values of the predicted_signal
% are 0. We could try to fill them out, though this would cause a additive
% error in the next calculations.
for i = length(a)+1 : n
    for j = 1 : length(a)
        predicted_signal(i) = predicted_signal(i) + a(j) * y_cap_tone(i-j);
    end
end
end
end

```

d) DPCM_encoder:

```

function [y,y_cap_tone,output_signal,predicted_signal,a] =
DPCM_encoder(input_signal,p,N,min_value,max_value)
% DPCM_encoder inputs:
% 1. y: The number of previous values of the signal we want to keep.
% 2. y_cap_tone: processed output signal.
% 3. output_signal: the signal that gets transmitted to the receiver.
% 4. predicted_signal: the predicted signal.
% 5. a: the coefficients of the prediction of the signal.
%
% DPCM_encoder outputs:
% 1. input_signal: the signal that we want to transmit.
% 2. p: the number of coefficients we want.
% 3. N,min_value,max_value: See my_quantizer.m

% We initialize the vectors.
[~,~,a] = prediction_filter(p,input_signal);
predicted_signal = zeros(size(input_signal));
y = zeros(size(input_signal));
output_signal = zeros(size(input_signal));
y_cap_tone = zeros(size(input_signal));

% We implement the given functions of the exercise.
for i = 1: length(input_signal)
    y(i) = input_signal(i) - predicted_signal(i);
    [output_signal(i),~] = my_quantizer(y(i),N,min_value,max_value);
    y_cap_tone(i) = output_signal(i) + predicted_signal(i);
    predicted_signal = predictor(a, length(input_signal),y_cap_tone);
end
end

```

e) DPCM_decoder:

```

function [output_received_signal,predicted_received_signal] =
DPCM_decoder(received_signal,quantized_coeff)
% DPCM_decoder inputs:
% 1. received_signal: The signal that gets outputted at the transmitter
end.
% 2. quantized_coeff: The coefficients that get quantized at the
transmitter end and sent to the receiver beforehand..
%
% DPCM_decoder outputs:
% 1. output_received_signal: The signal that gets outputted at the
receiver end.
% 2. predicted_received_signal: The output of the predictor at the
receiver end.

```

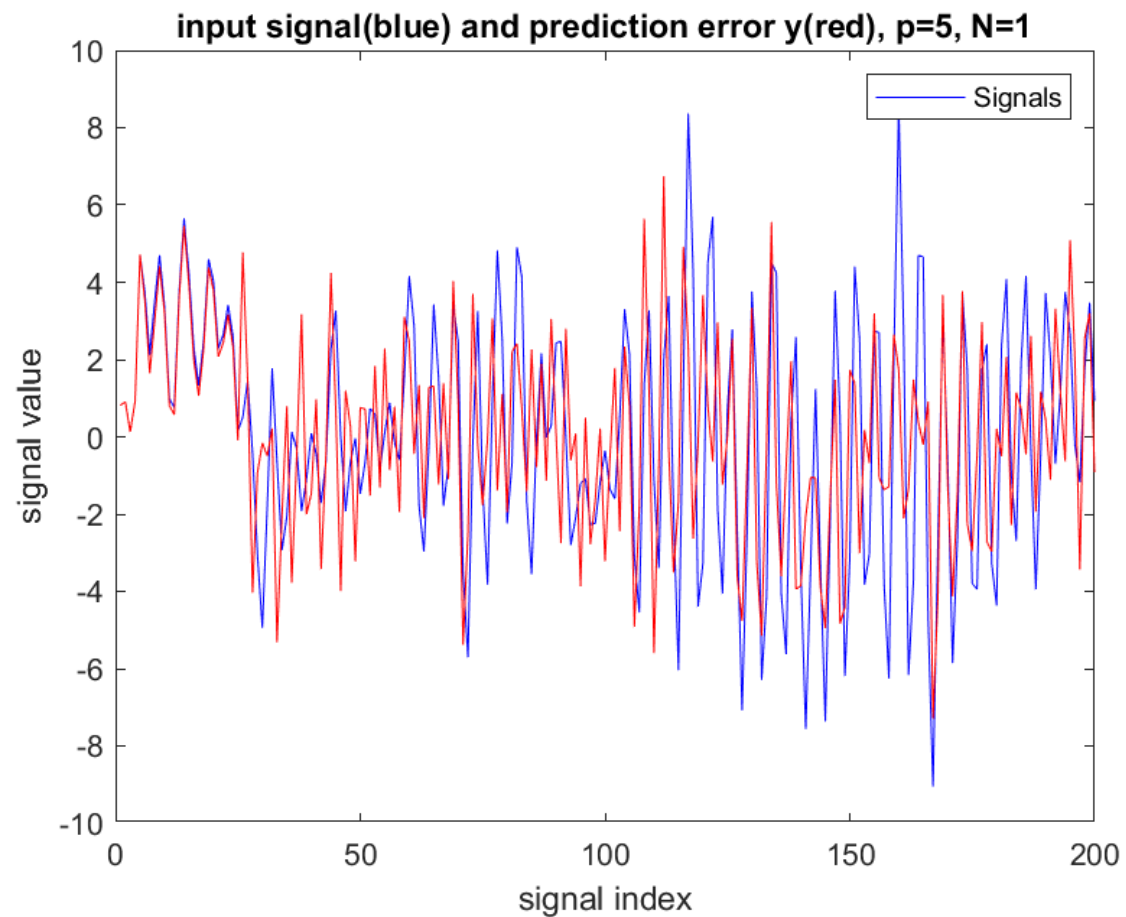
```

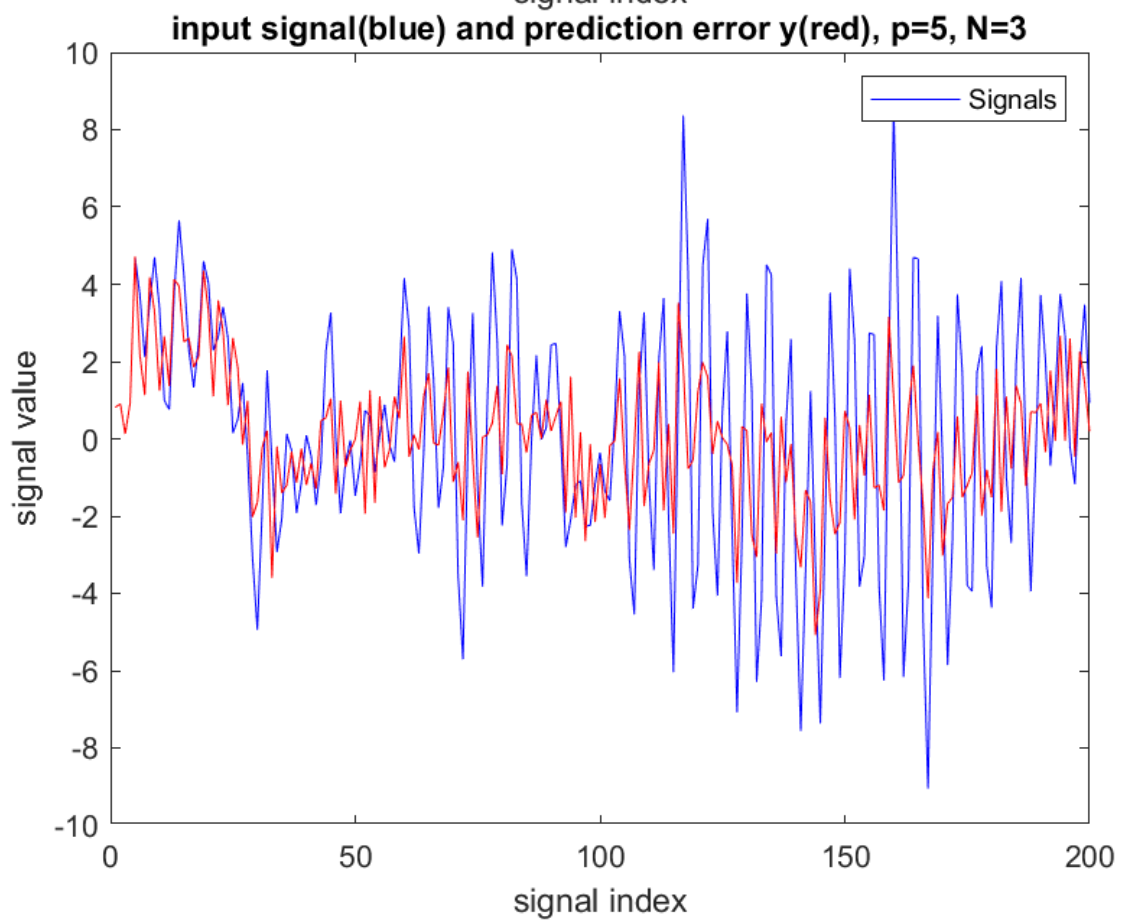
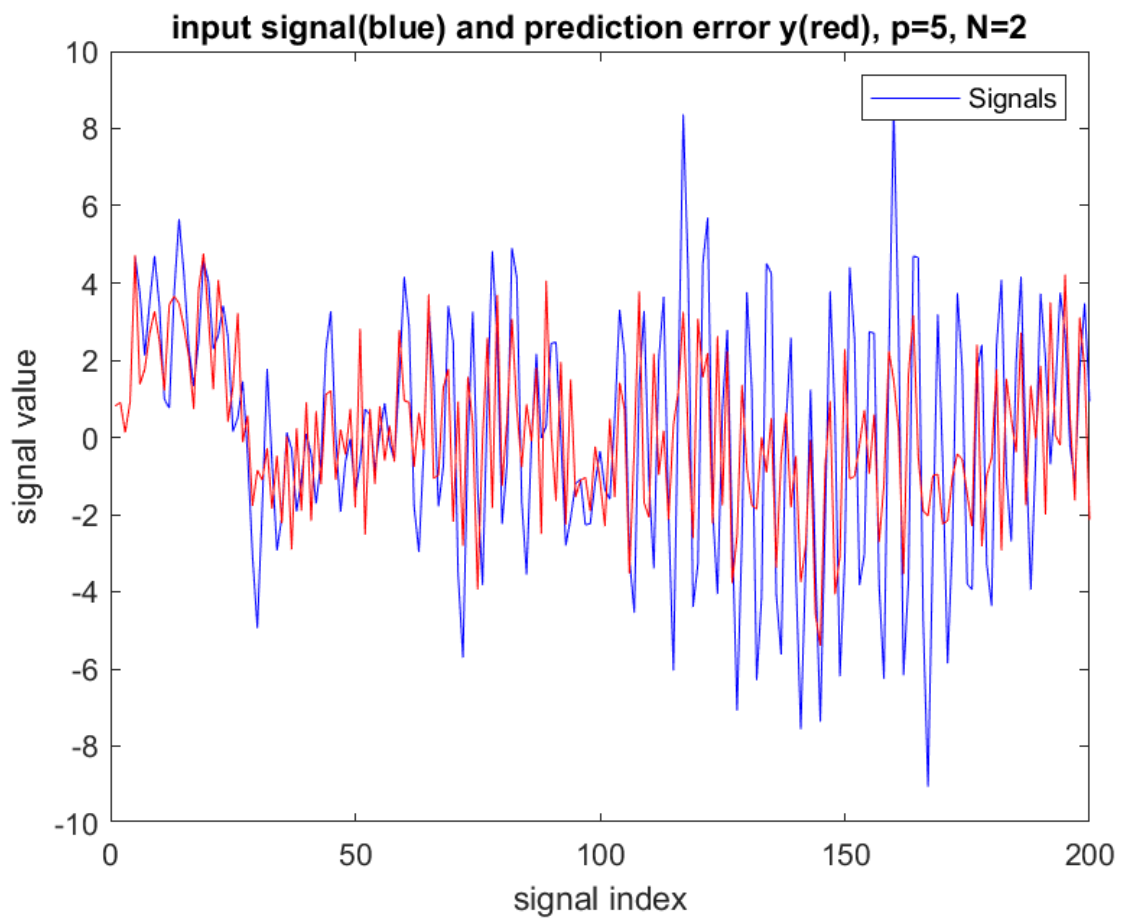
output_received_signal = zeros(size(received_signal));
predicted_received_signal = zeros(size(received_signal));

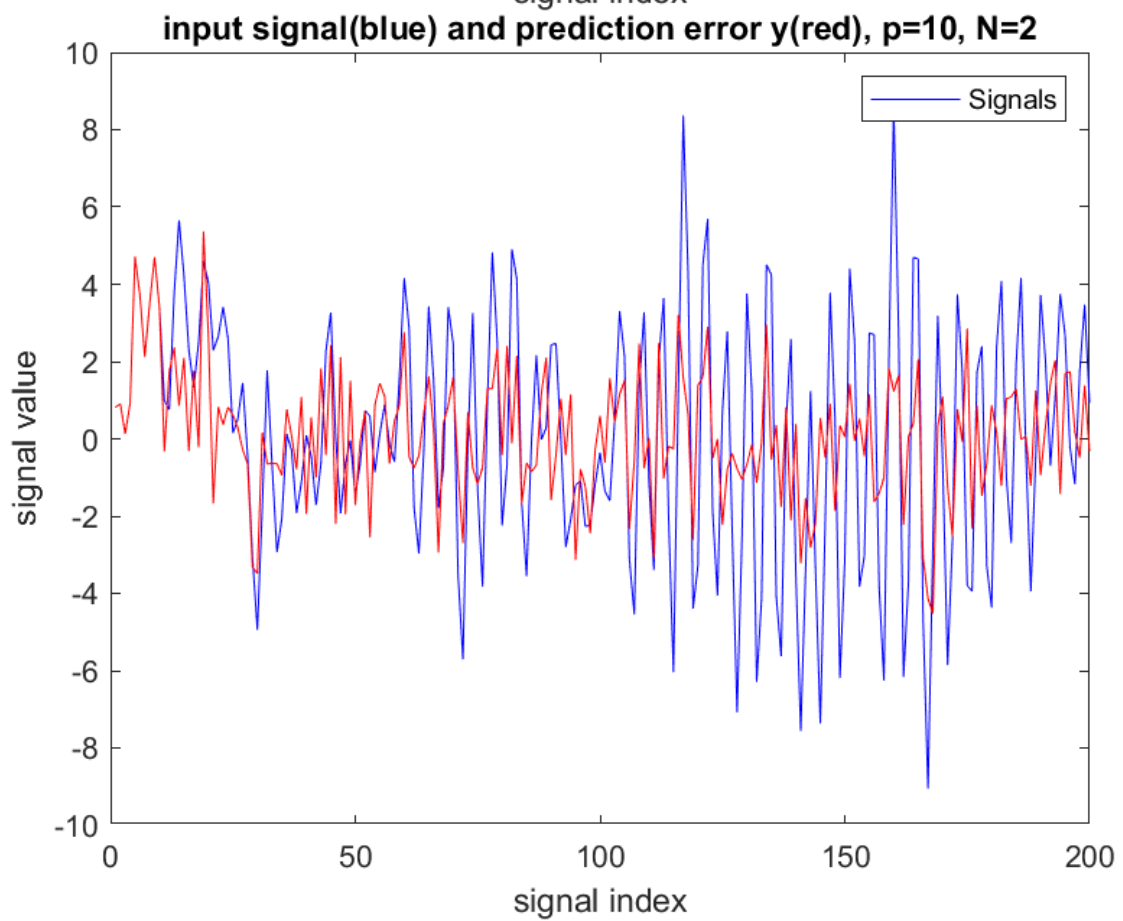
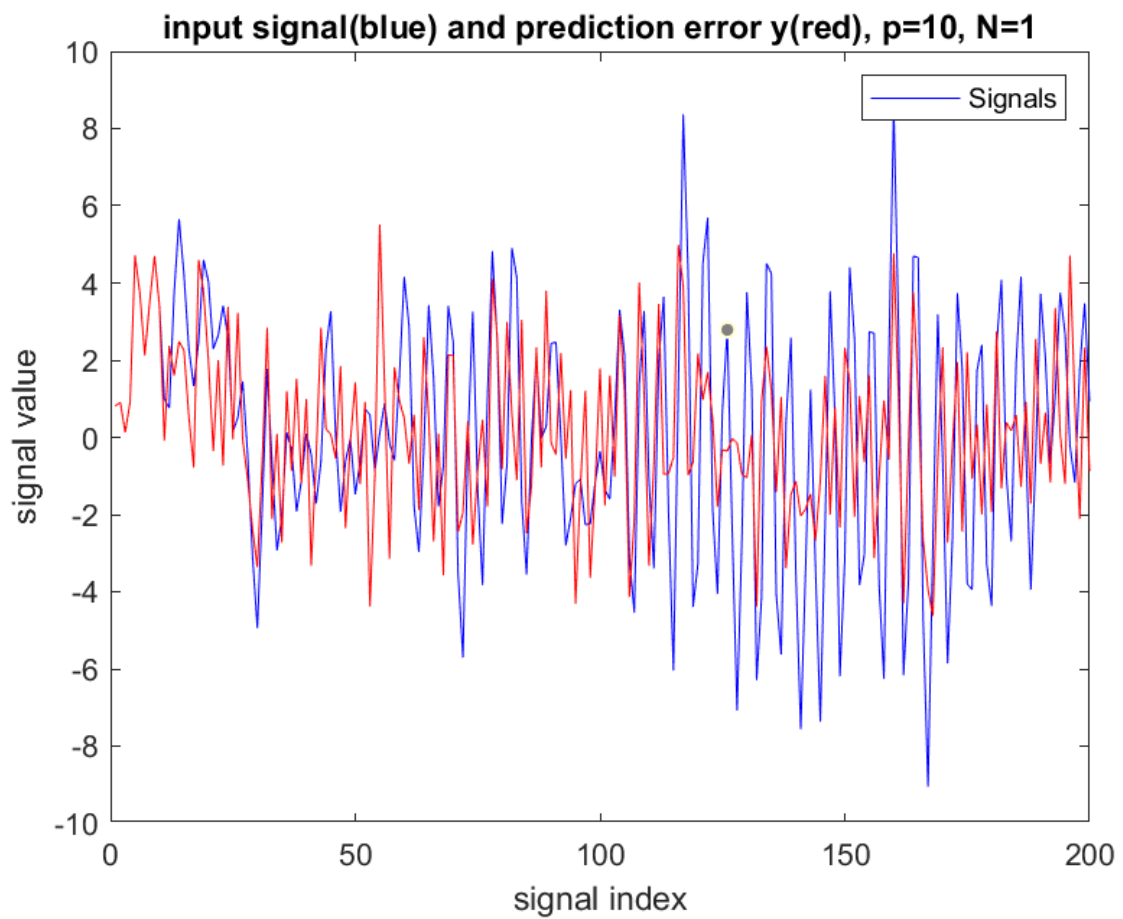
for i = 1 : length(received_signal)
    output_received_signal(i) = received_signal(i) +
predicted_received_signal(i);
    predicted_received_signal = predictor(quantized_coeff,
length(received_signal),output_received_signal);
end
end

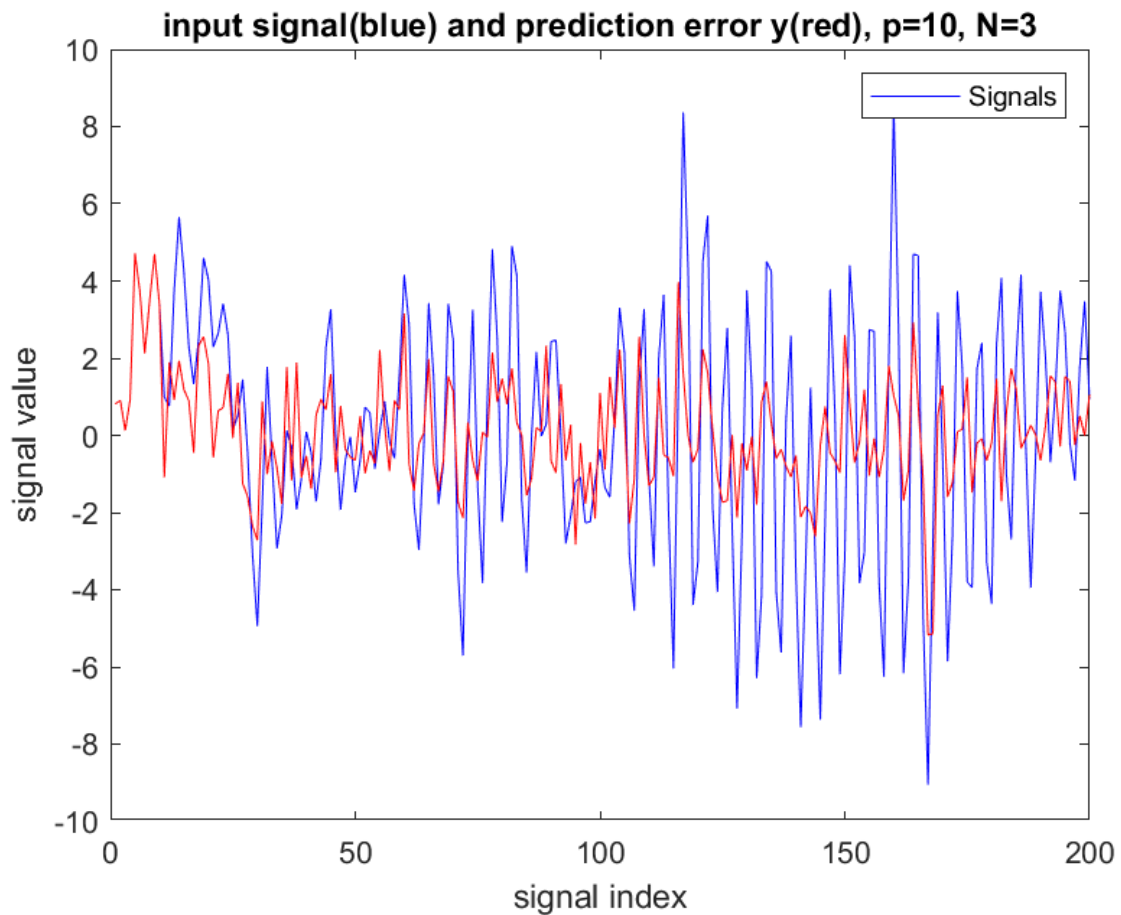
```

2. Επιλέγουμε 2 τιμές για το πλήθος των συντελεστών πρόβλεψης $p = 5, 10$. Έπειτα, χρησιμοποιώντας τον ακόλουθο κώδικα για την γραφική απεικόνιση των αποτελεσμάτων. Για θέματα ευκρίνειας, απεικονίζουμε μόνο τα πρώτα 200 δείγματα, μεταξύ του σήματος εισόδου (input_signal) και του σφάλματος πρόβλεψης(y). Τα αποτελέσματα φαίνονται παρακάτω:









Σημασία του p:

Παρατηρούμε ότι όσο αυξάνεται το p , δηλαδή όσο μεγαλώνει ο αριθμός των συντελεστών πρόβλεψης, το σφάλμα πρόβλεψης μικραίνει σε μέγεθος. Αυτό συμβαίνει διότι με μικρότερο αριθμό συντελεστών πρόβλεψης, ο προβλεπτής του συστήματος μας αδυνατεί να κατανοήσει το σήμα, συνεπώς οι τιμές που παράγει διαφέρουν αρκετά από το σήμα εισόδου, μεγαλώνοντας το σφάλμα πρόβλεψης.

Αυξάνοντας τους συντελεστές πρόβλεψης, το σφάλμα πρόβλεψης, όπως φαίνεται και από τις γραφικές παραστάσεις, μειώνεται συνειδητά, καθώς με περισσότερους συντελεστές, ο προβλεπτής μπορεί να μοντελοποιήσει το σήμα με περισσότερη ακρίβεια. Ωστόσο η τεράστια αύξηση του αριθμού των συντελεστών, μπορεί να οδηγήσει σε προβλήματα Overfitting, ιδιαίτερα σε σήματα που έχουν λιγότερη προβλεψιμότητα. Συνεπώς, η τακτική αυτή δεν θα μεγαλώσει πολύ το performance, ωστόσο θα οδηγήσει σε πολύ μεγαλύτερη υπολογιστική πολυπλοκότητα.

Σημασία του N:

Το N , όπως μας ζητείται, αναπαριστά τον αριθμό των δυαδικών ψηφίων για τον κβαντισμό του σήματος. Συνεπώς για N ψηφία, έχουμε 2^N στάθμες κβαντισμού. Όταν το N παίρνει μικρές τιμές, και κατά συνέπεια οι στάθμες κβαντισμού είναι λίγες, η ακρίβεια της αναπαράστασης του κάθε δείγματος του σήματος μειώνεται, μεγαλώνοντας έτσι την απόσταση μεταξύ τους, δίνοντας έτσι μεγαλύτερο σφάλμα κβάντισης, που είναι κομμάτι του σφάλματος πρόβλεψης. Συνεπώς, το σήμα σφάλματος πρόβλεψης θα αποκλίνει περισσότερο από το σήμα εισόδου.

Αυξάνοντας τον αριθμό των δυαδικών ψηφίων, ωστόσο, μεγαλώνει την ακρίβεια του κβαντισμού του σήματος, εφόσον με περισσότερες στάθμες κβάντισης, προσεγγίζεται καλύτερα η ακριβής τιμή του σήματος, μειώνοντας έτσι το σφάλμα κβάντισης, αφήνοντας έτσι το σφάλμα πρόβλεψης να εξαρτάται κυρίως από τις δυνατότητες του προβλεπτή.

Finetuning:

Η ισορροπία μεταξύ του αριθμού των συντελεστών πρόβλεψης και του αριθμού των δυαδικών ψηφίων για τον κβαντισμό είναι ιδιαίτερης σημασίας για την αποδοτικότερη λειτουργία του συστήματος μας. Όπως βλέπουμε και από τα αποτελέσματά μας, η αύξηση του αριθμού των συντελεστών πρόβλεψης, δεν είναι πάντα η καλύτερη λύση, εάν ο θόρυβος του κβαντισμού είναι μεγαλύτερος, ενώ για πολύ μεγάλες τιμές του, μπορεί να είναι υπολογιστικά απαιτητικό. Από την άλλη μεριά, η αύξηση του αριθμού των σταθμών κβάντισης, μόνο βοηθητικό είναι στην ποιότητα του σήματος, φτάνοντας στο σημείο που το σφάλμα που παράγεται από την διαδικασία της πρόβλεψης να είναι ο μόνος περιοριστικός παράγοντας.

3. Παρακάτω φαίνεται ο κώδικας που υλοποιήθηκε για το συγκεκριμένο ερώτημα:

```
N_values = 1:3;
p_values = 5:10;
mse_results = zeros(length(p_values), length(N_values));

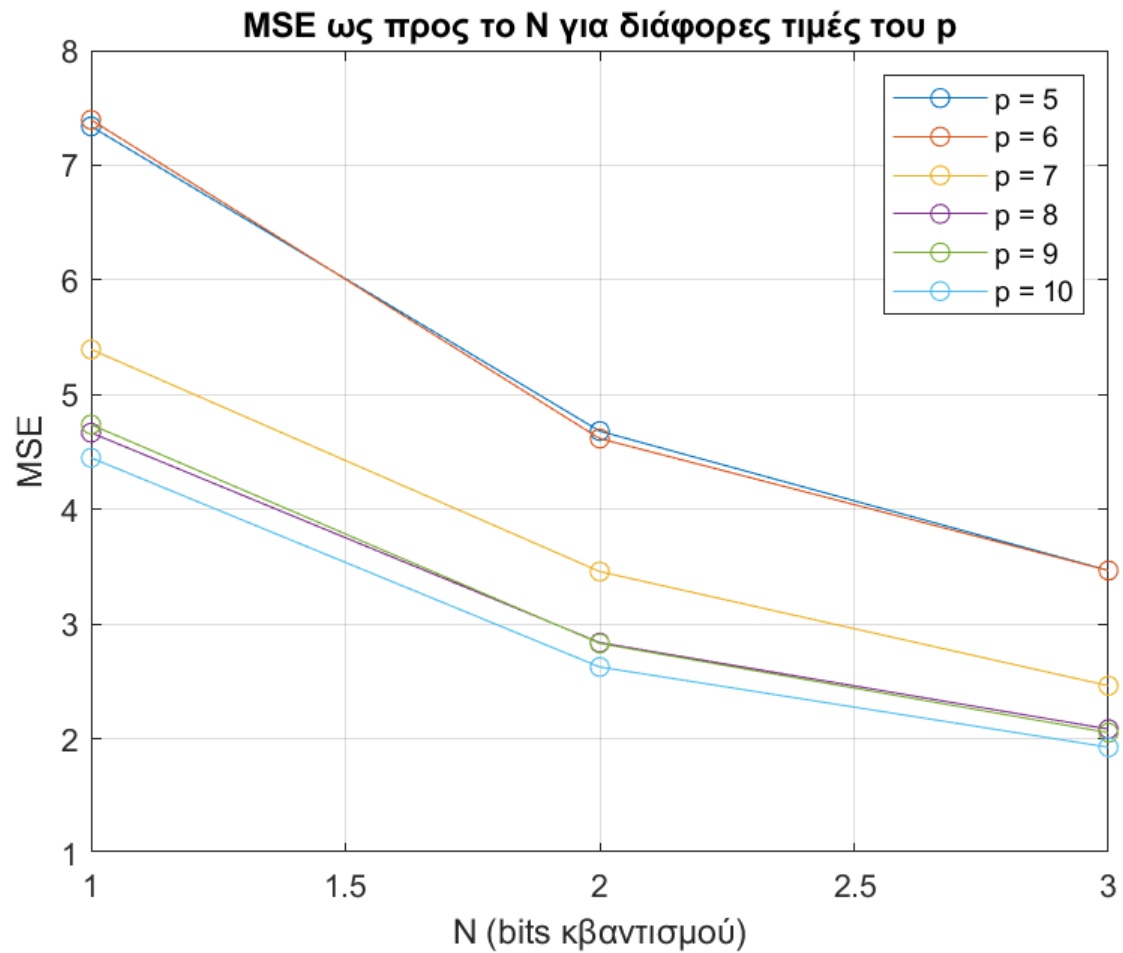
for i = 1:length(p_values)
    p = p_values(i);
    for j = 1:length(N_values)
        N = N_values(j);

        [~,~,~,predicted_signal,~] = DPCM_encoder(input_signal,p,N, -
3.5,3.5);

        %MSE
        mse_results(i, j) = immse(input_signal, predicted_signal);
    end
end

% graph
figure;
plot(N_values, mse_results', '-o');
title('MSE ως προς το N για διάφορες τιμές του p');
xlabel('N (bits κβαντισμού)');
ylabel('MSE');
legend(arrayfun(@(p) sprintf('p = %d', p), p_values, 'UniformOutput',
false));
grid on;
```

Παρακάτω, φαίνεται το διάγραμμα που προέκυψε, καθώς και οι συντελεστές πρόβλεψης για κάθε συνδυασμό των p, N :



$p = 5$, Coefficients: 1.2852, -1.5856, 0.9901, -0.5424, -0.0287

$p = 6$, Coefficients: 1.2865, -1.5626, 0.9481, -0.4750, -0.0834, 0.0425

$p = 7$, Coefficients: 1.2650, -1.5207, 1.1885, -0.9547, 0.7069, -0.6081, 0.5056

$p = 8$, Coefficients: 1.0876, -1.3073, 0.9403, -0.6198, 0.2901, -0.0751, 0.0622, 0.3505

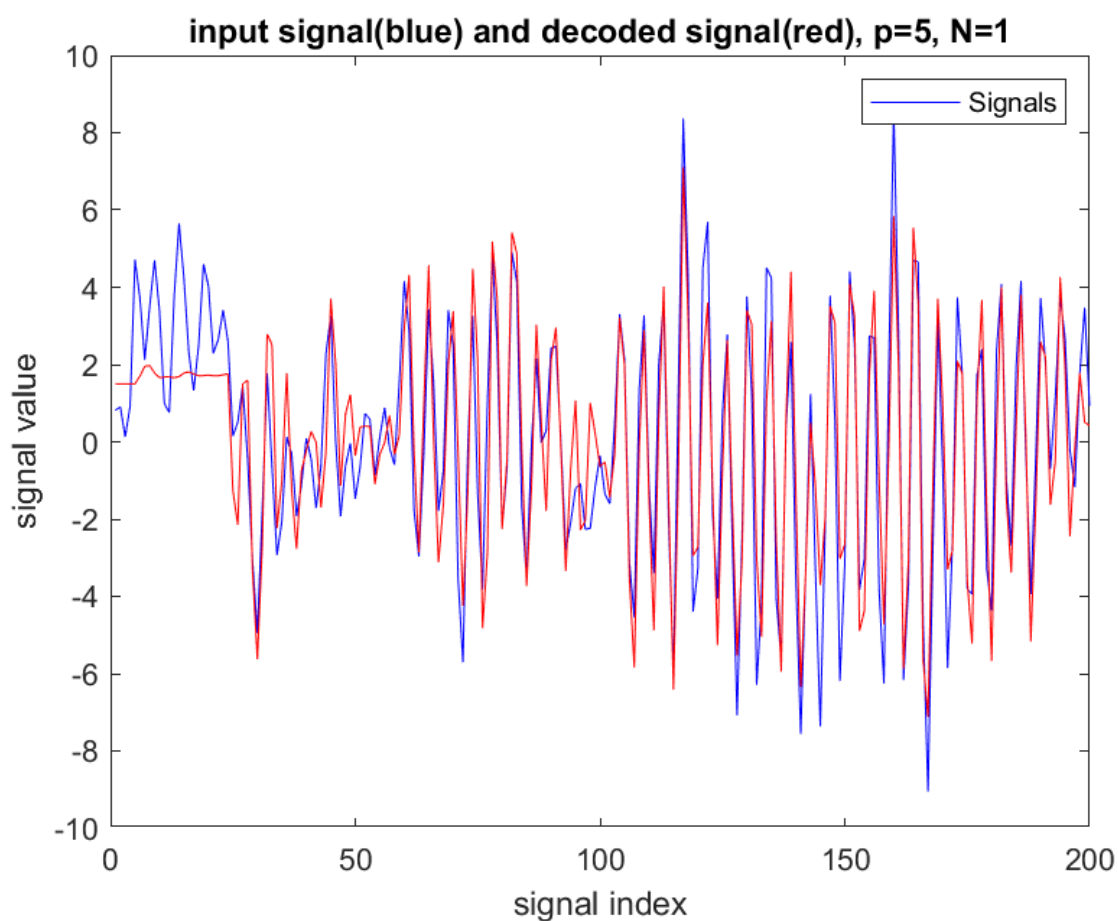
$p = 9$, Coefficients: 1.1432, -1.2975, 0.9285, -0.5740, 0.1919, 0.0741, -0.1455, 0.5233, -0.1588

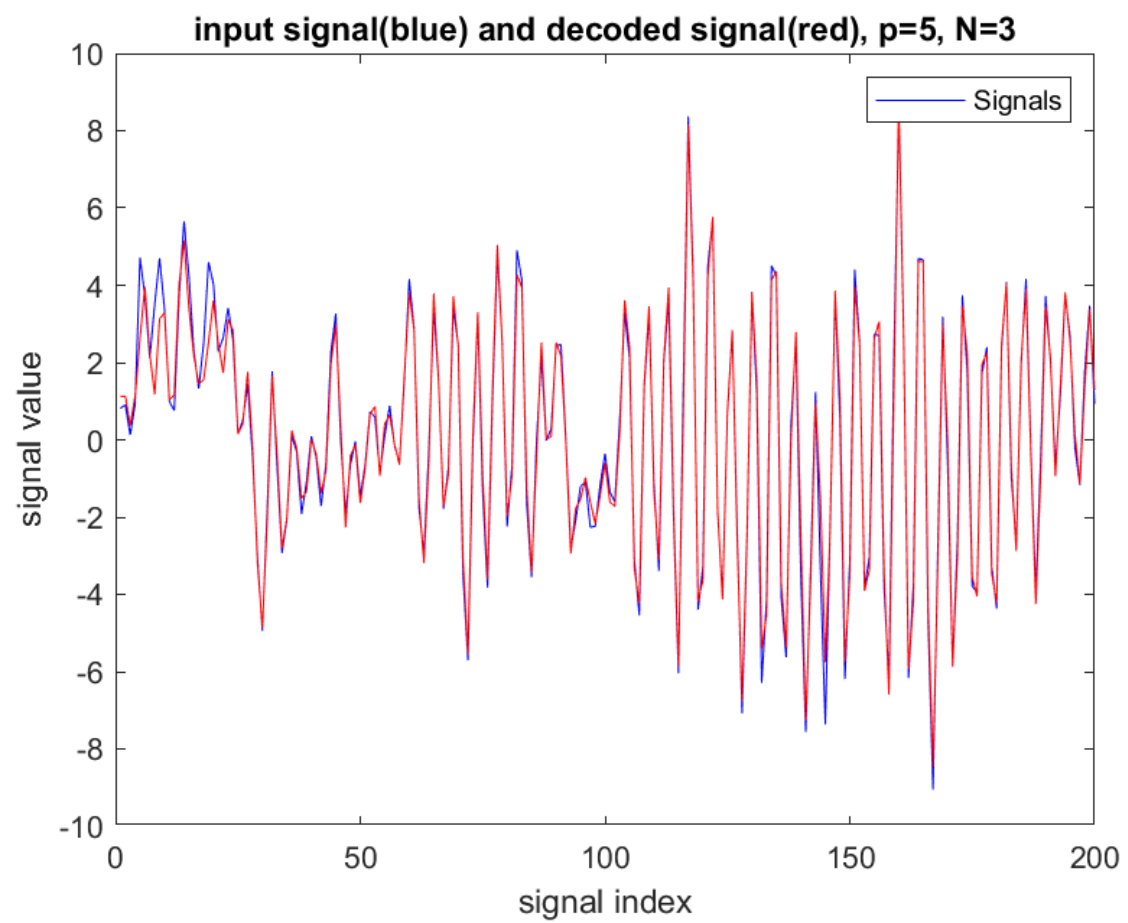
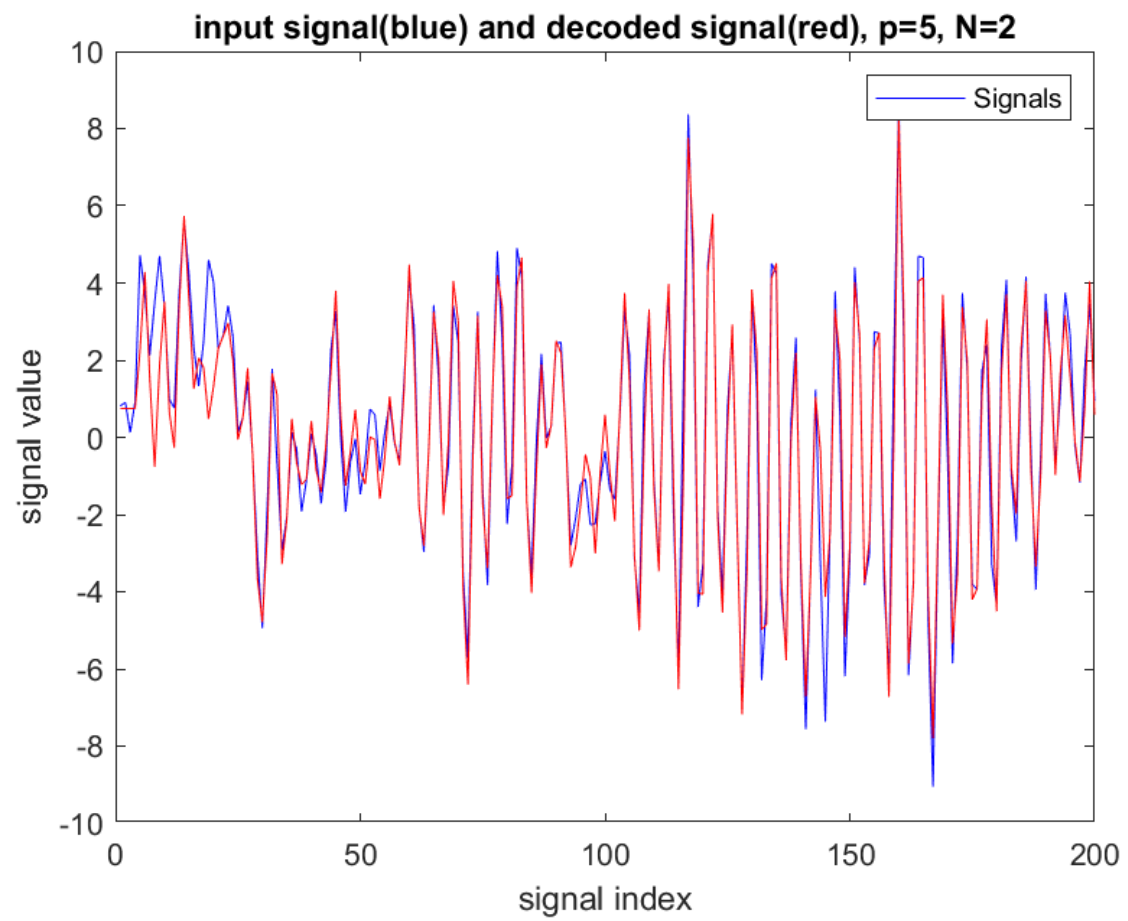
$p = 10$, Coefficients: 1.1063, -1.1763, 0.8947, -0.5567, 0.2361, -0.0585, 0.0694, 0.2230, 0.1058, -0.2314

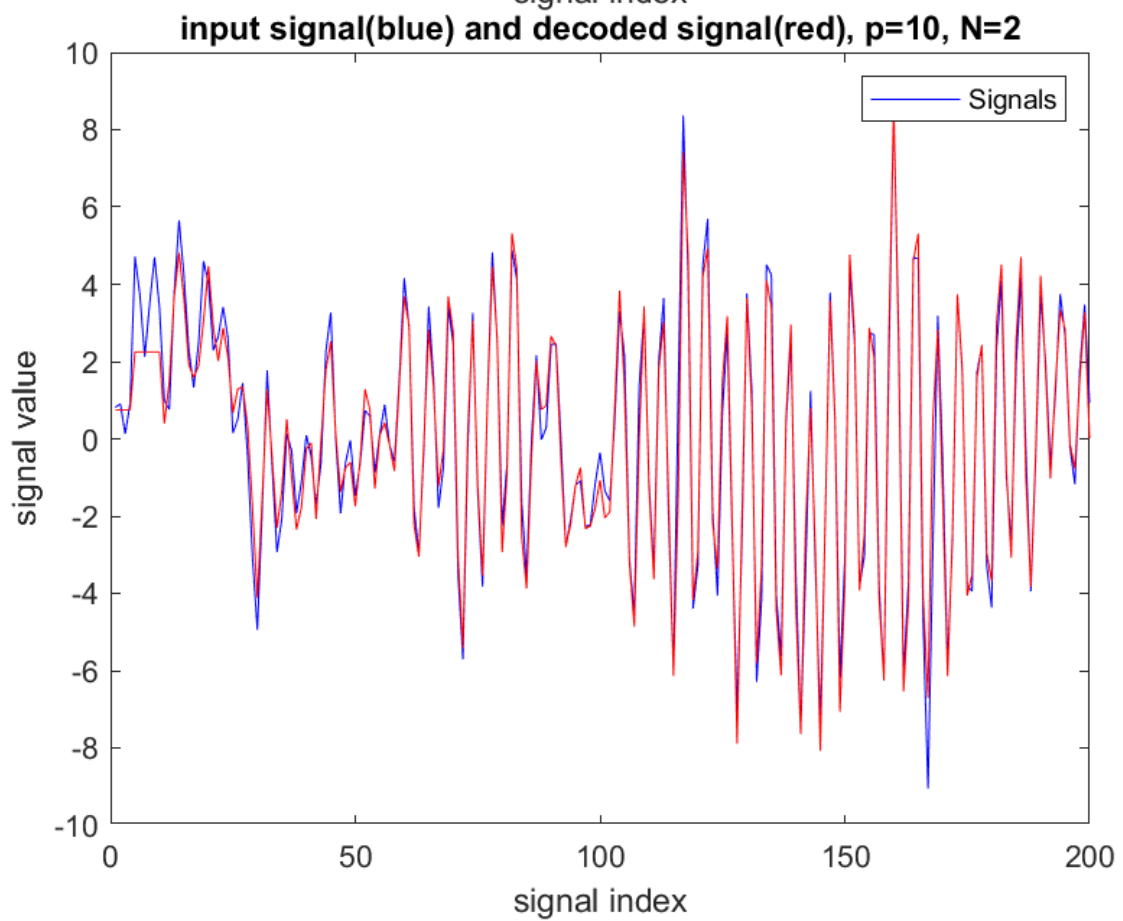
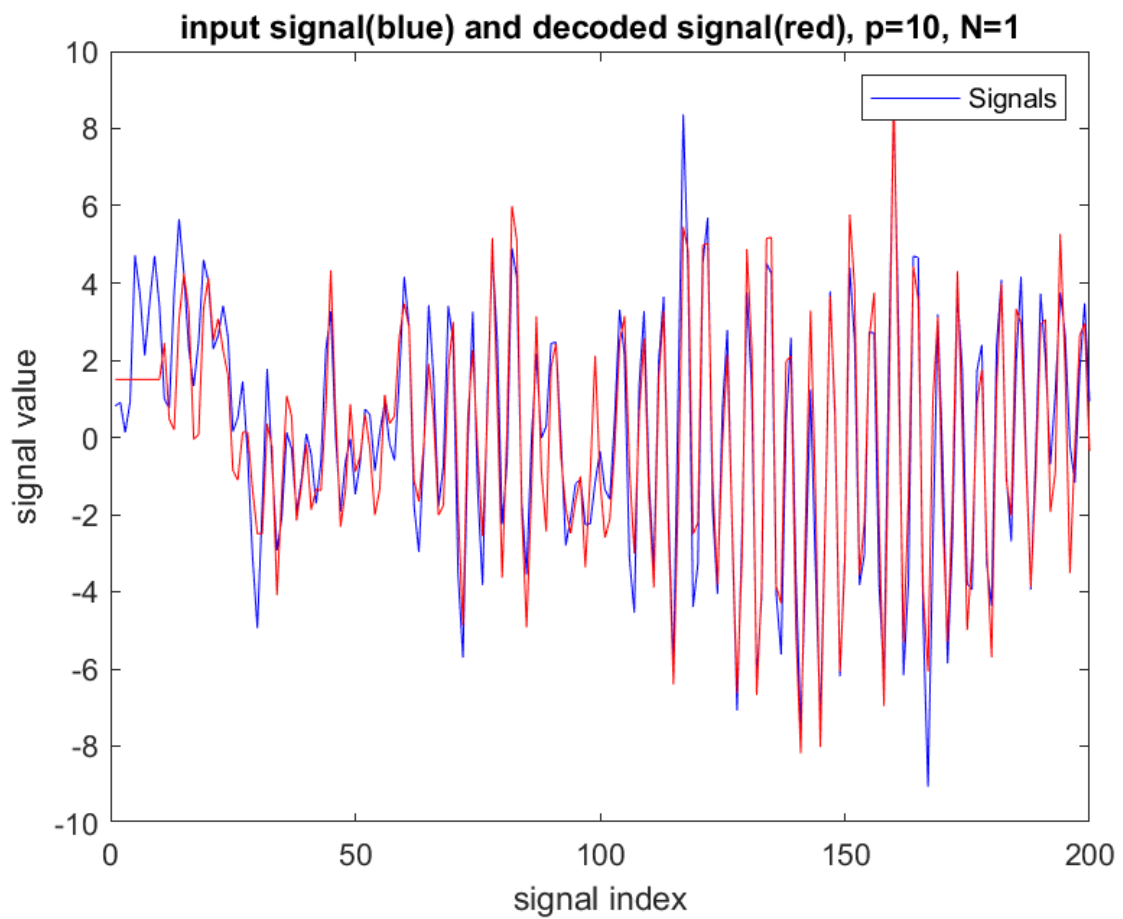
Από το διάγραμμα παρατηρούμε, ότι ειπώθηκε και παραπάνω, ότι με την αύξηση των δυαδικών ψηφίων για την κβάντιση, το μέσο τετραγωνικό σφάλμα μεταξύ του προβλεπόμενου σήματος και του σήματος εισόδου μειώνεται.

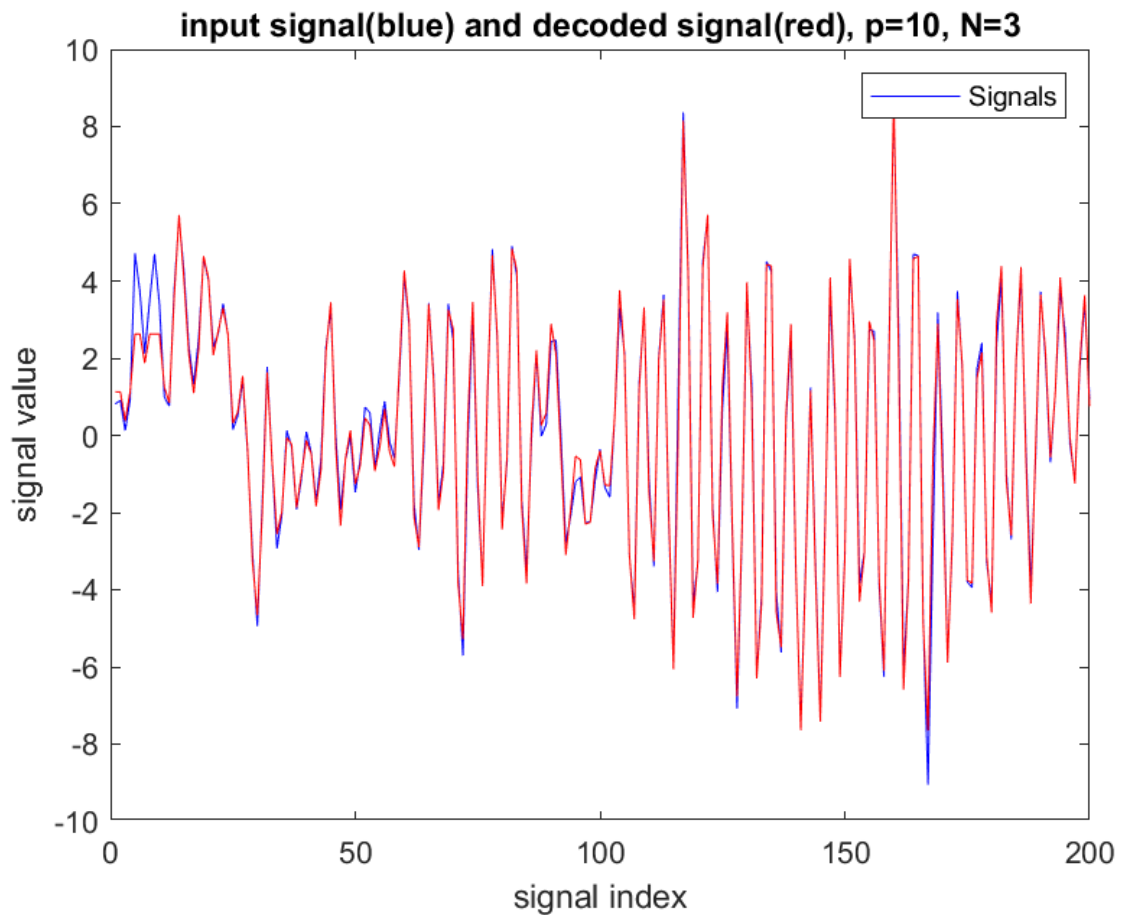
Ωστόσο αυξάνοντας τον αριθμό των συντελεστών πρόβλεψης, ιδιαίτερα για μεγάλες τιμές του p , δεν παρατηρείται μεγάλη βελτίωση από την μία τιμή στην άλλη. Αυτό μπορεί να οφείλεται σε διάφορους παράγοντες, όπως στην συμπεριφορά του σήματος. Δεδομένου ότι το σήμα είναι αρκετά προβλέψιμο, οι επιπλέον παράγοντες πρόβλεψης δεν συμμετέχουν σημαντικά στον καθορισμό των προβλεπόμενων τιμών του.

4. Για την κβάντιση των συντελεστών πρόβλεψης, χρειάζεται η ξεχωριστή τους κβάντιση, χρησιμοποιώντας την συνάρτηση `my_quantizer`, που υλοποιήσαμε. Με αυτό υπόψιν, παρακάτω φαίνεται τόσο το αρχικό, όσο και το ανακατασκευασμένο σήμα, για τα πρώτα 200 δείγματα (για λόγους ευκρίνειας):





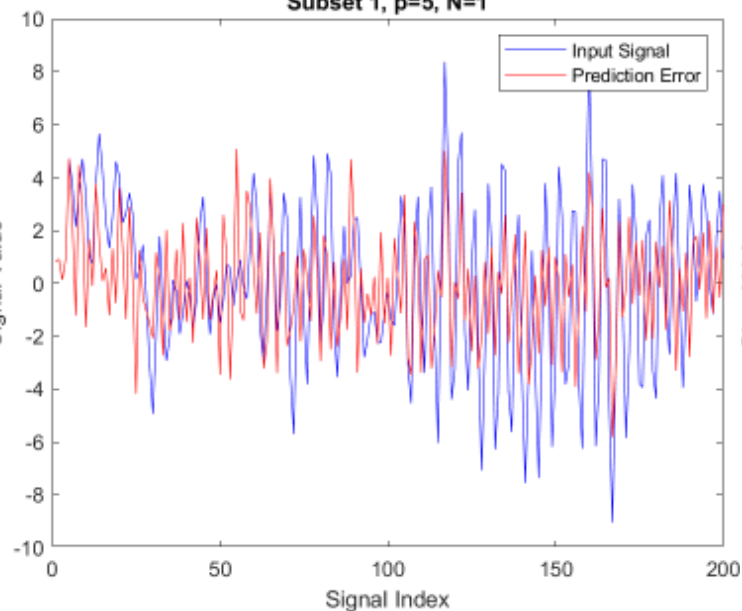




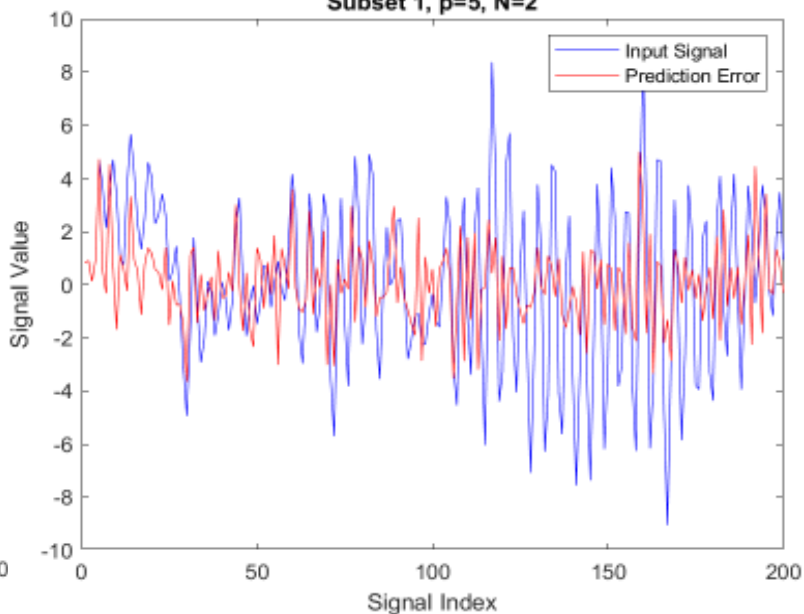
Όπως παρατηρούμε και από τα διαγράμματα, τόσο η αύξηση του αριθμού των συντελεστών πρόβλεψης (p), όσο και του αριθμού δυαδικών ψηφίων για κβάντιση, οδηγεί σε πιο ακριβή αναπαράσταση του σήματος. Ωστόσο σε ένα πραγματικό σύστημα, ο μεγάλος αριθμός συντελεστών πρόβλεψης μπορεί να οδηγήσει, όπως προ είπαμε, σε φαινόμενα υπερπροσαρμογής, άρα και αύξησης της υπολογιστικής πολυπλοκότητας του συστήματος, ενώ η κβάντιση πολλών επιπέδων είναι και πιο λεπτομερής, ωστόσο απαιτεί περισσότερο bandwidth στο κανάλι μέσα από το οποίο θα μεταφερθεί. Συνεπώς, καταλαβαίνουμε ότι υπάρχει ένα trade-off ανάμεσα στην ακρίβεια της πρόβλεψης και την αποφυγή της υπερπροσαρμογής, καθώς και ανάμεσα στην ποιότητα της κβάντισης και την απαιτούμενη ποσότητα δεδομένων για μετάδοση.

Τώρα, χωρίζουμε το `input_signal` σε ισομεγέθη μη επικαλυπτόμενα υποσύνολα των 5000 δειγμάτων, και στην συνέχεια παραθέτουμε τα πρώτα 200 δείγματα του κάθε υποσυνόλου, καθώς και του σφάλματος πρόβλεψής του. Παρακάτω, φαίνονται τα διαγράμματα:

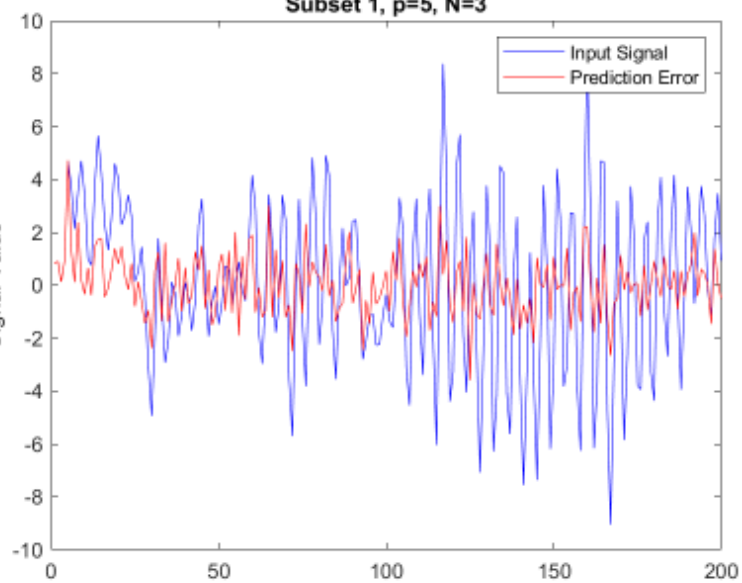
Subset 1, $p=5$, $N=1$



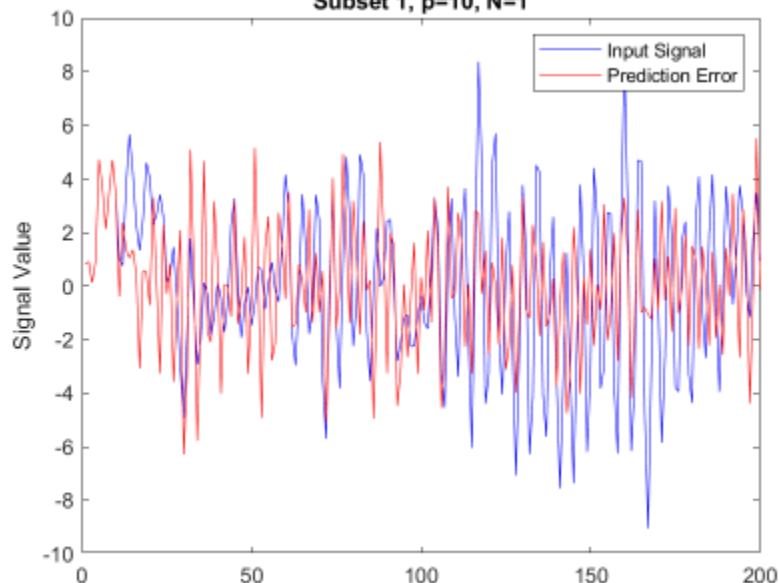
Subset 1, $p=5$, $N=2$



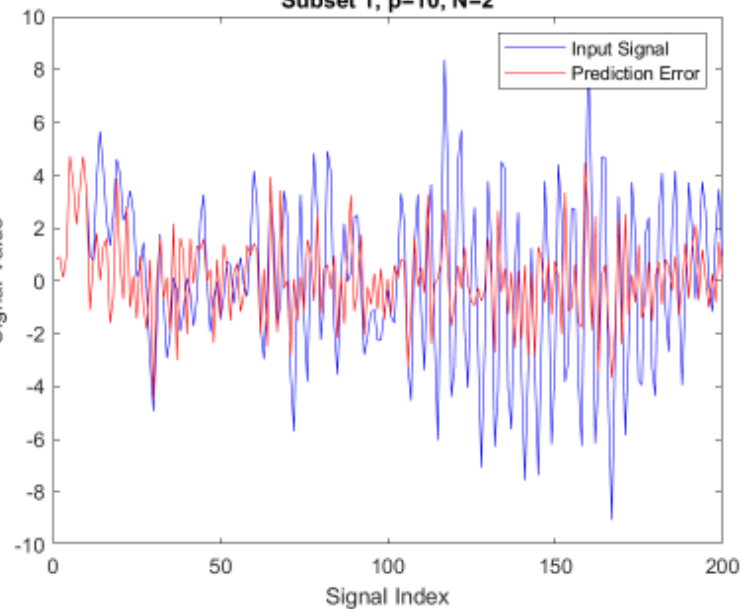
Subset 1, $p=5$, $N=3$



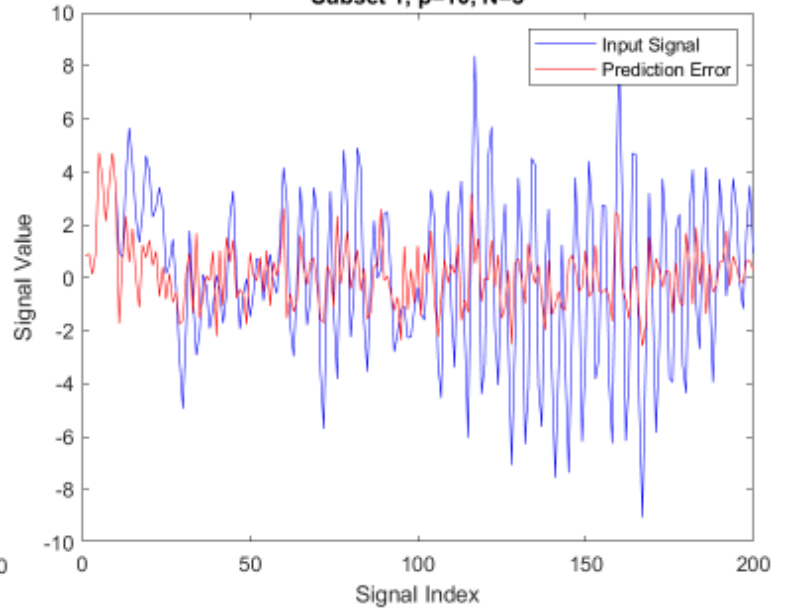
Subset 1, $p=10$, $N=1$

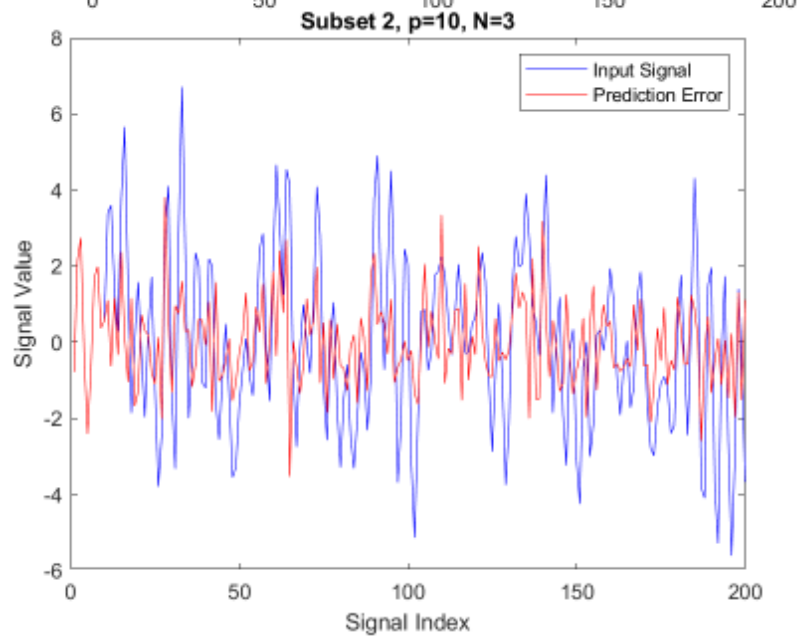
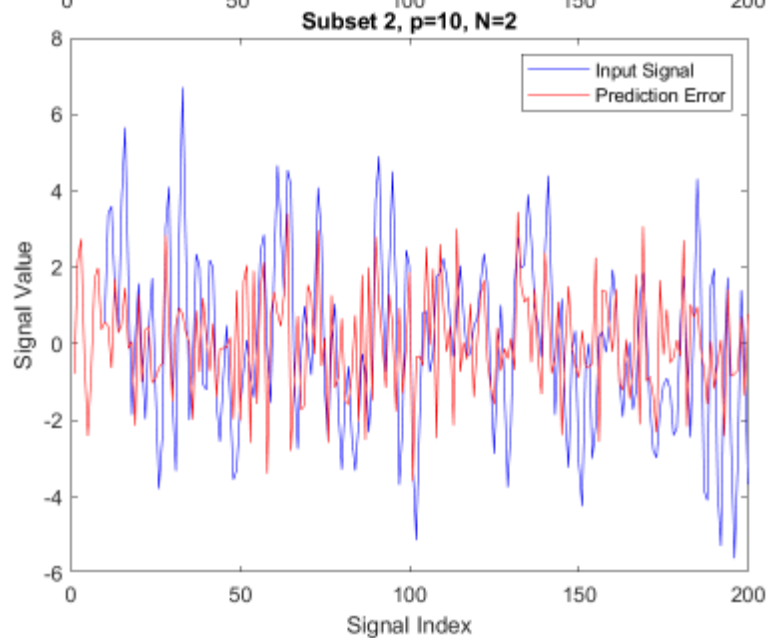
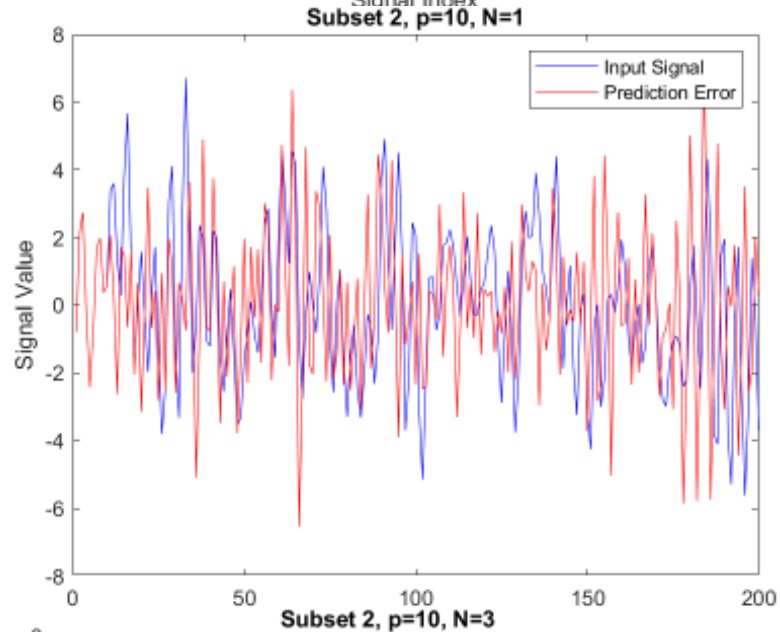
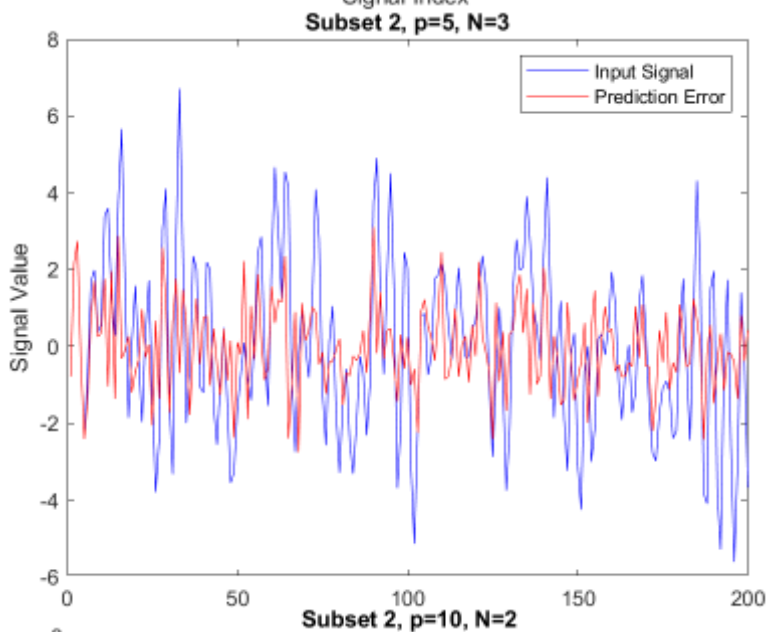
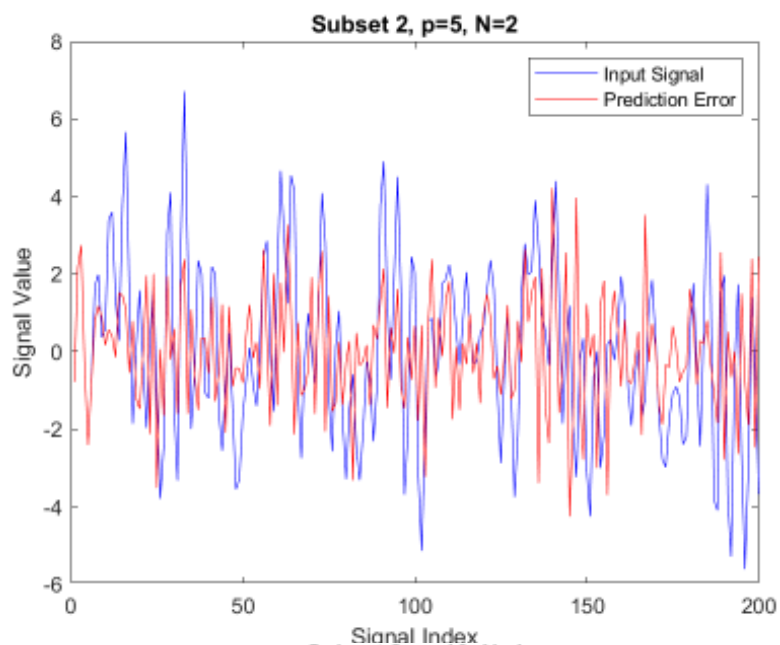
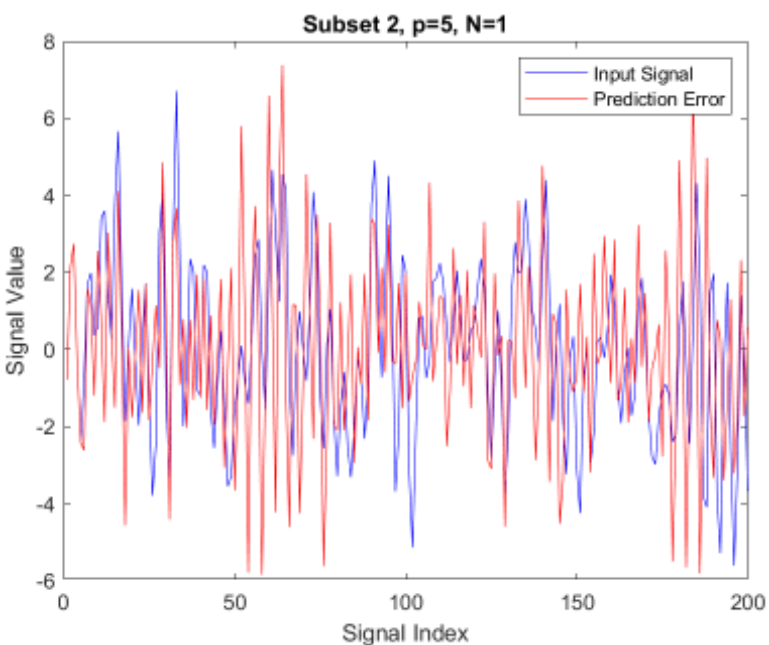


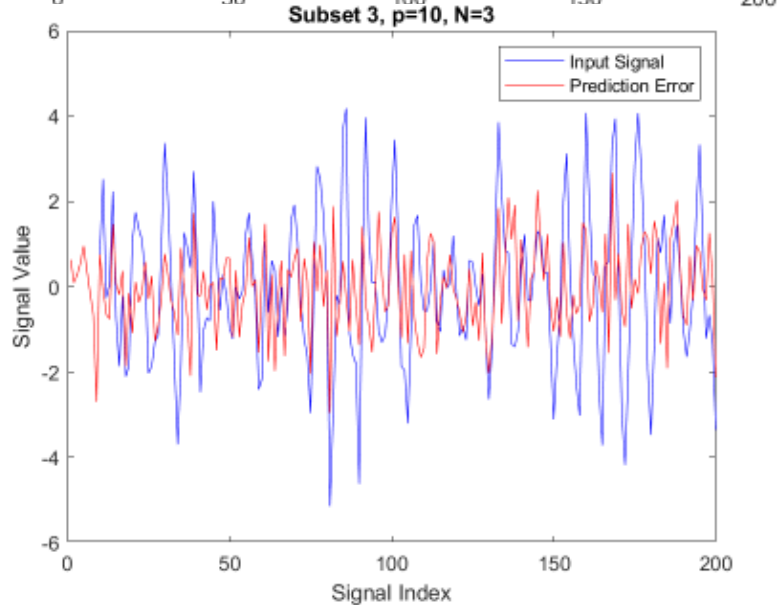
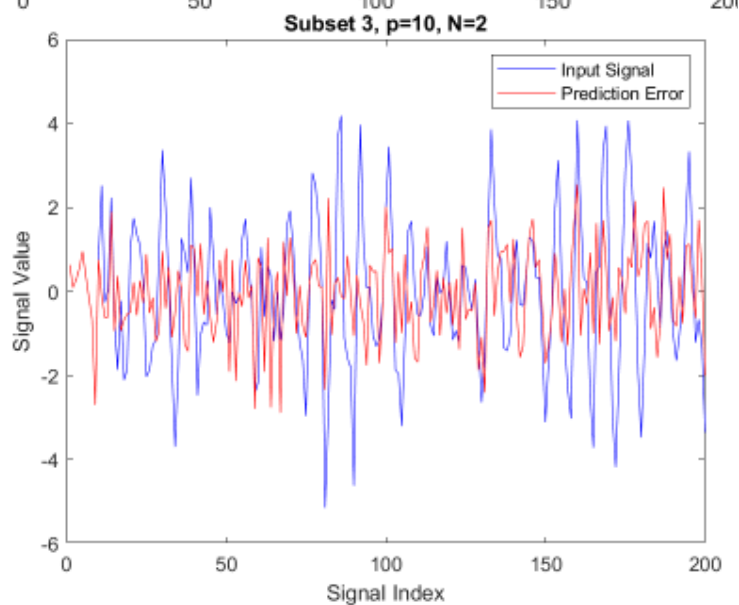
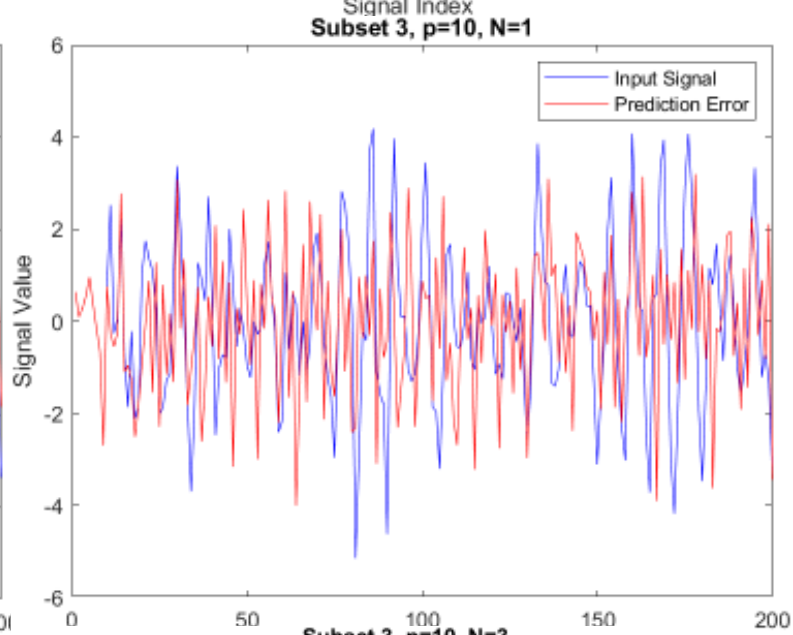
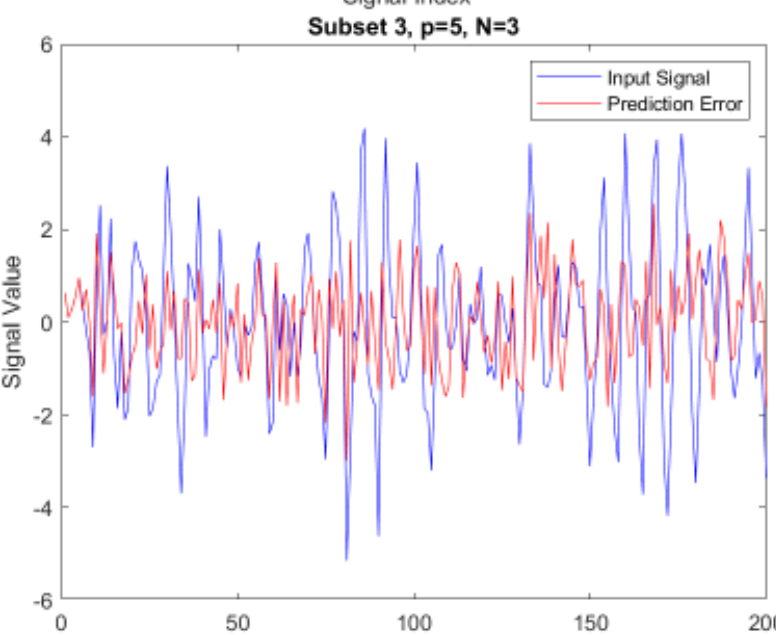
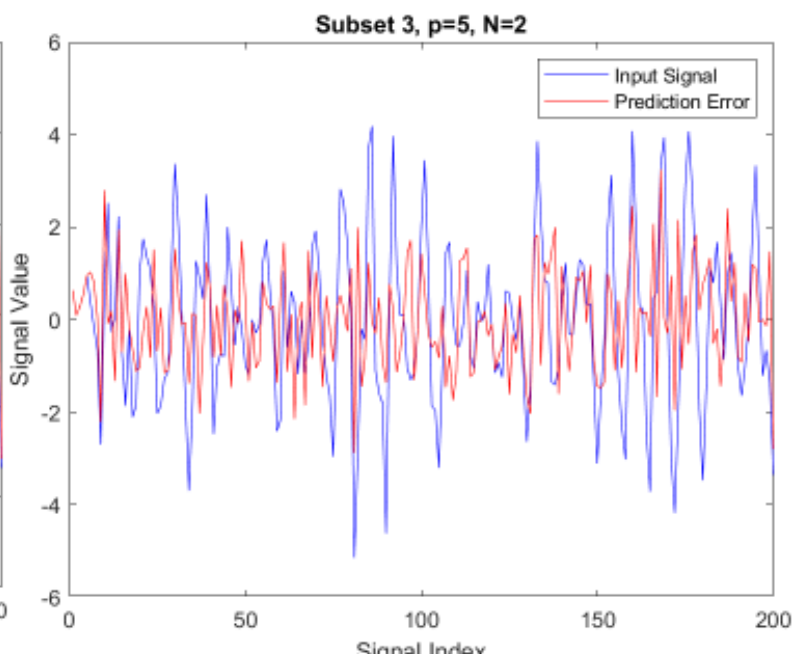
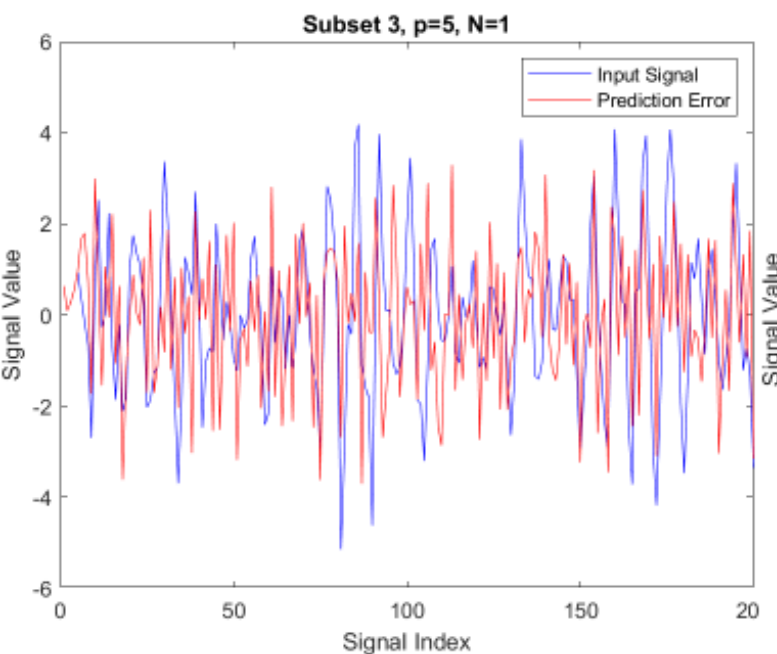
Subset 1, $p=10$, $N=2$

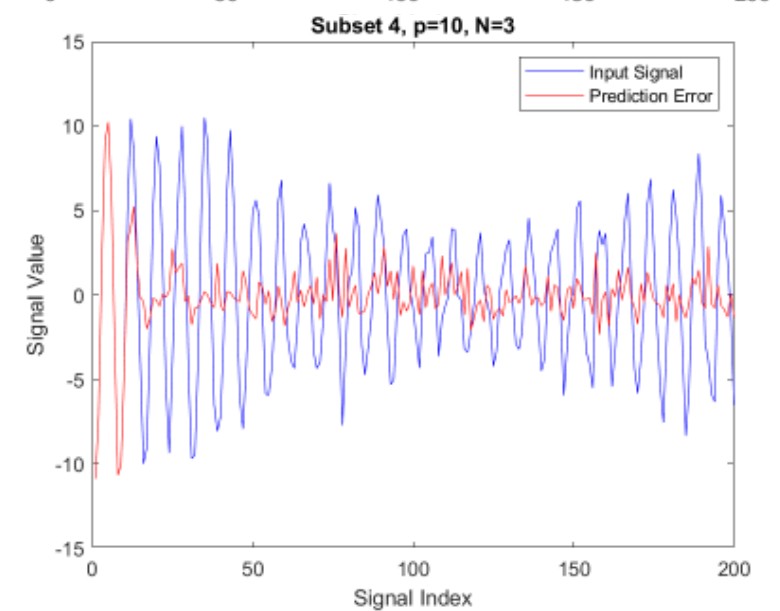
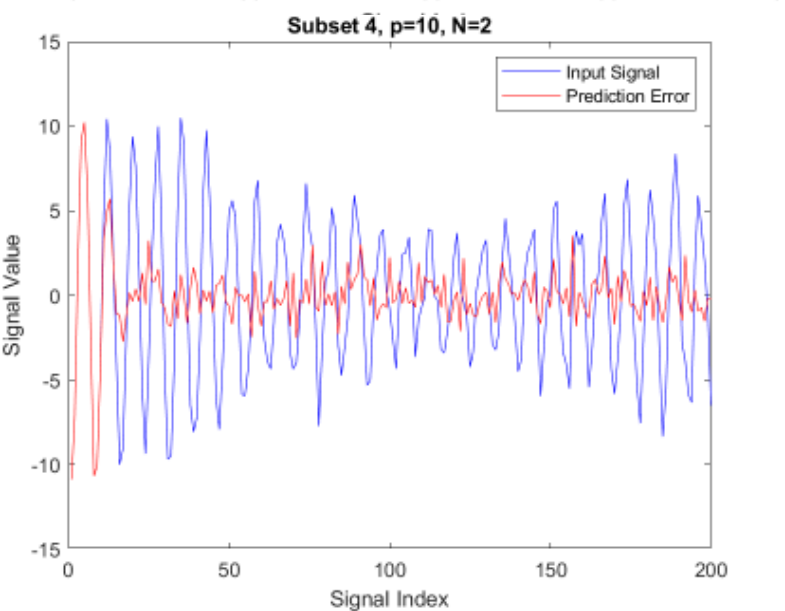
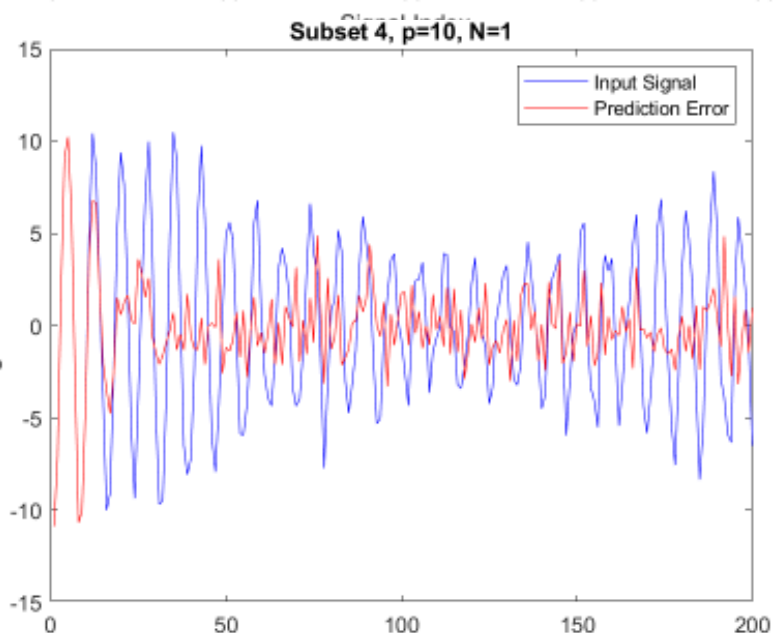
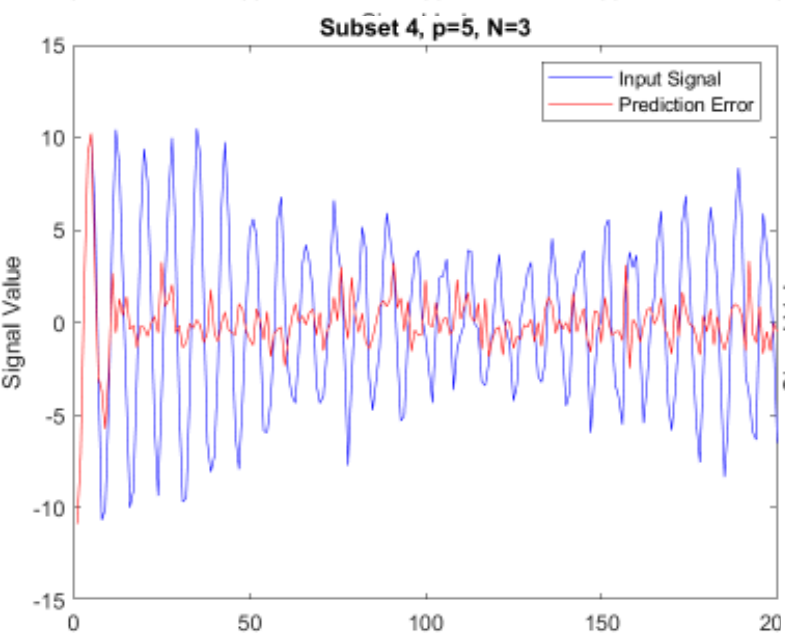
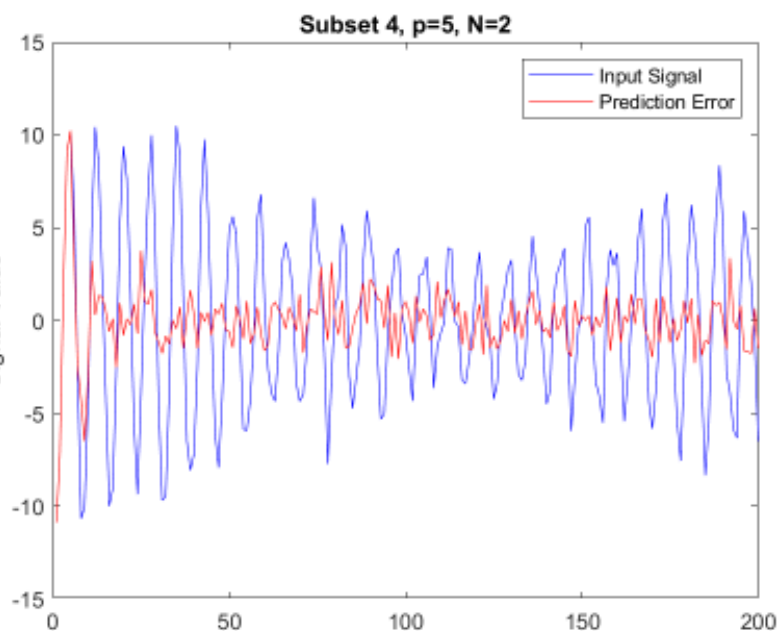
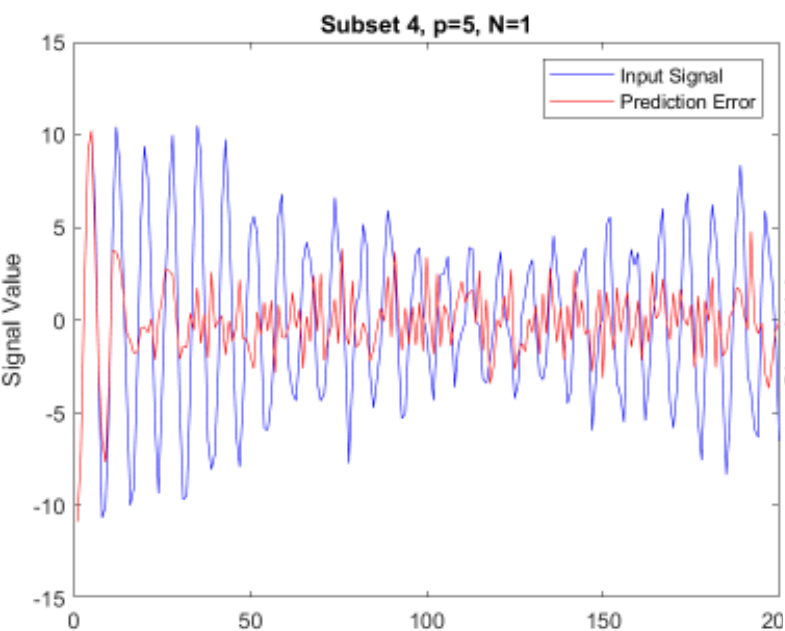


Subset 1, $p=10$, $N=3$

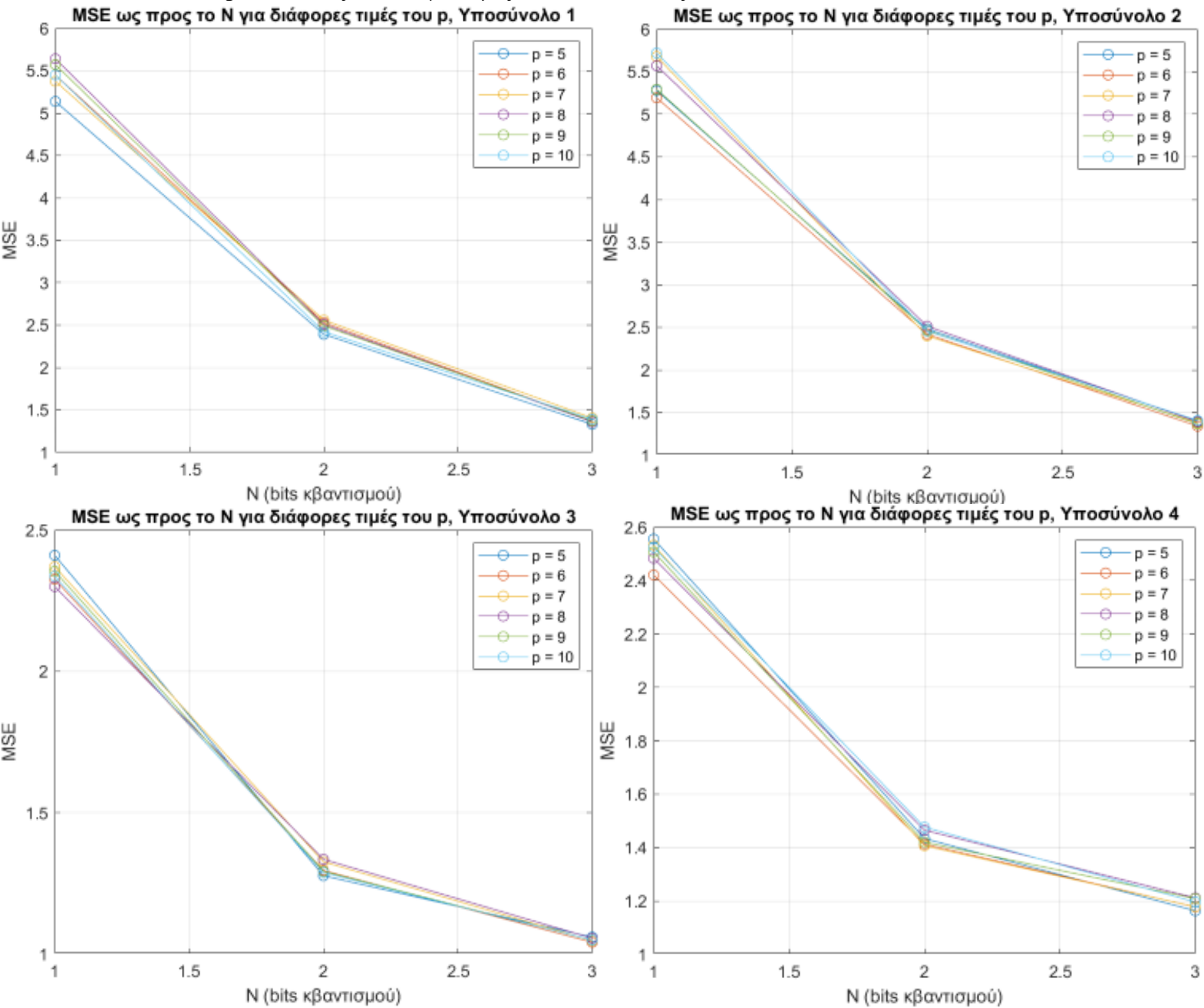




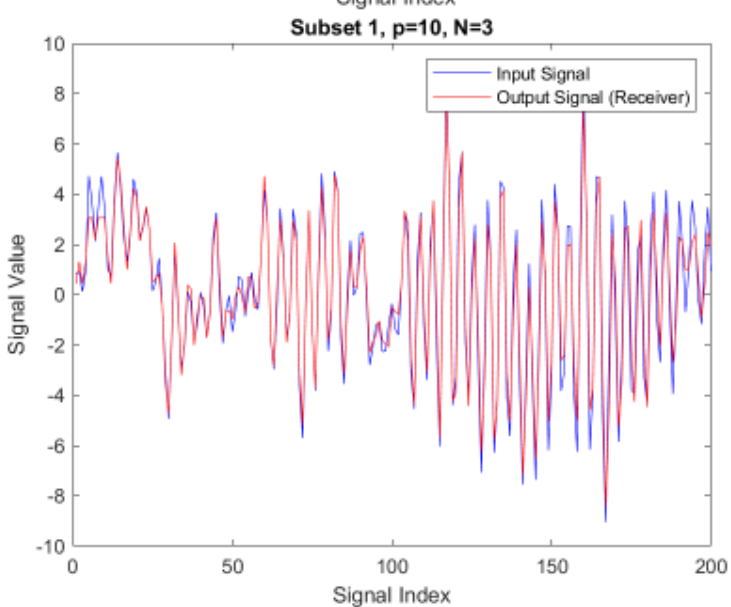
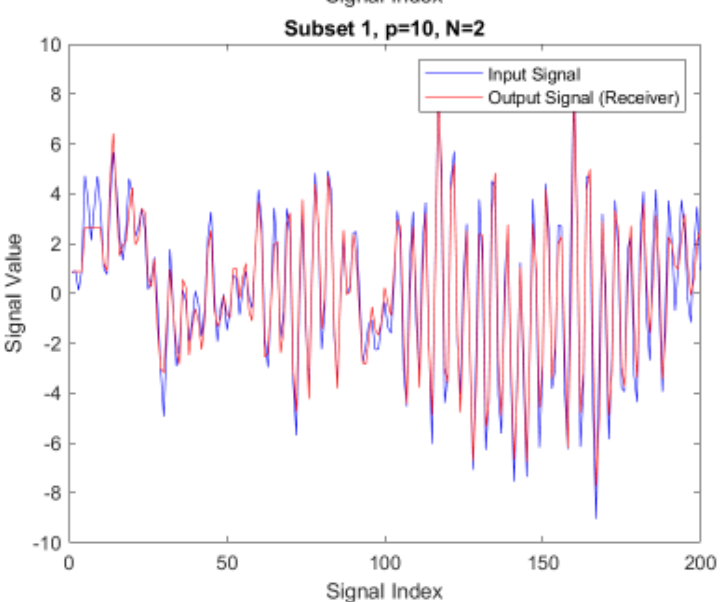
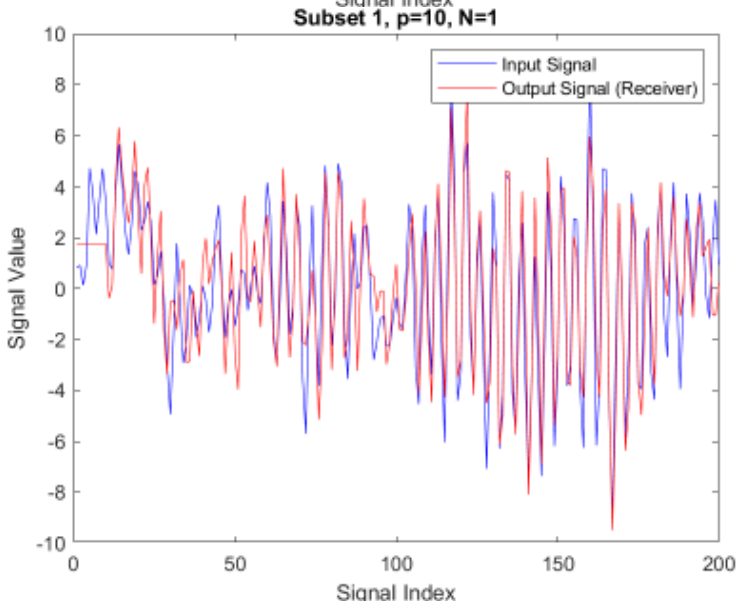
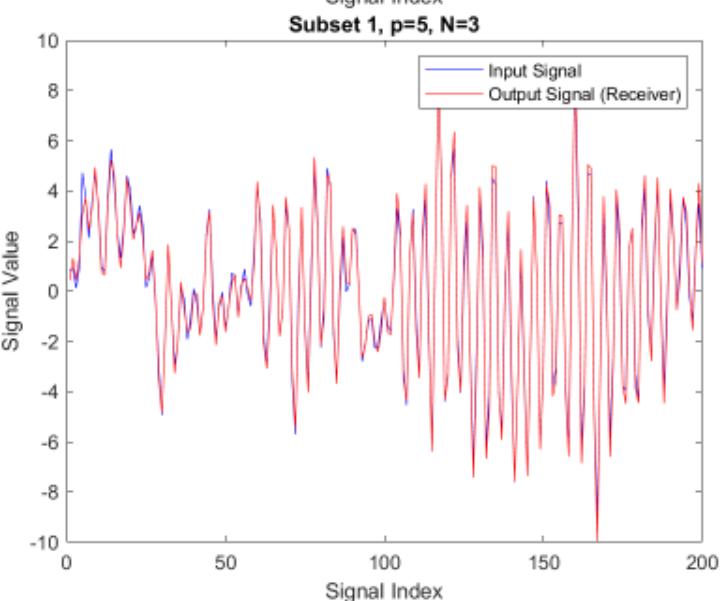
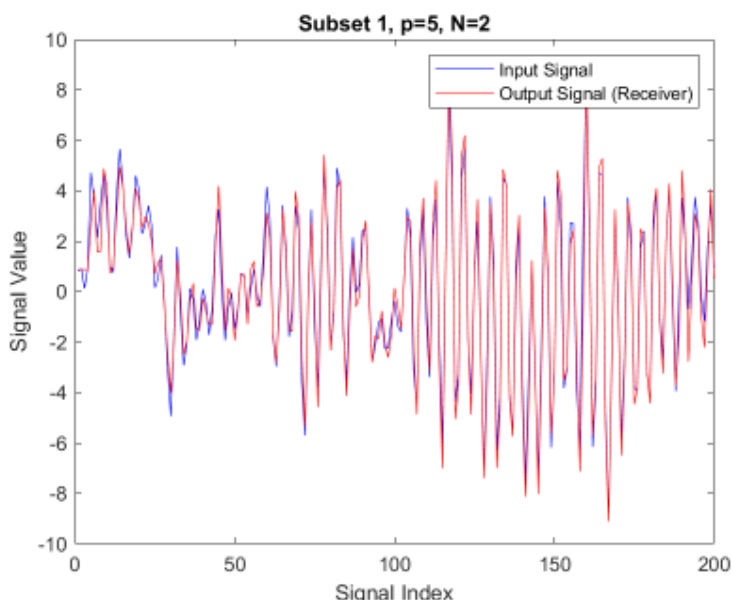
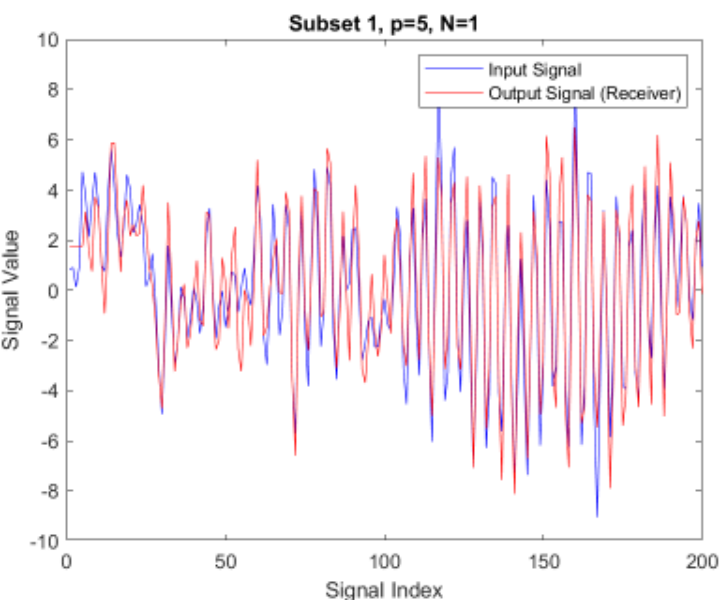




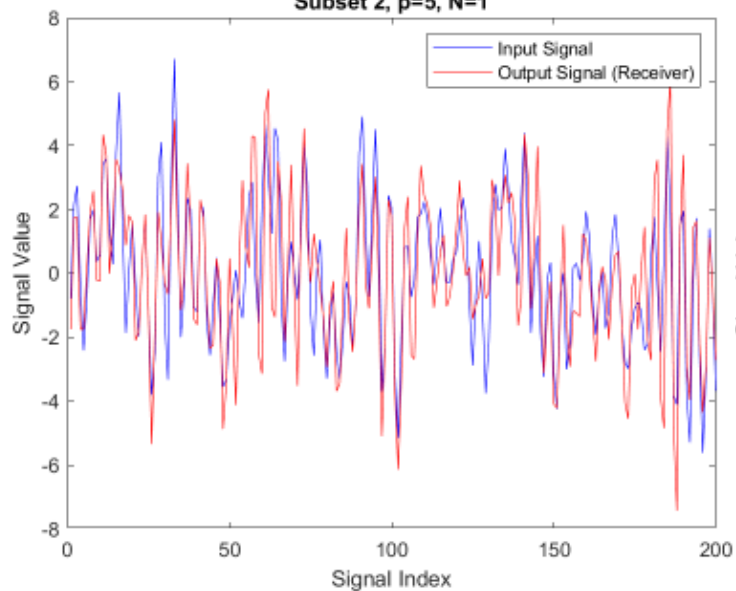
Παρακάτω, διακρίνουμε το μέσο τετραγωνικό σφάλμα κάθε υποσυνόλου για τιμές του p από 5 έως 10 και για τιμές του N από 1 έως 3:



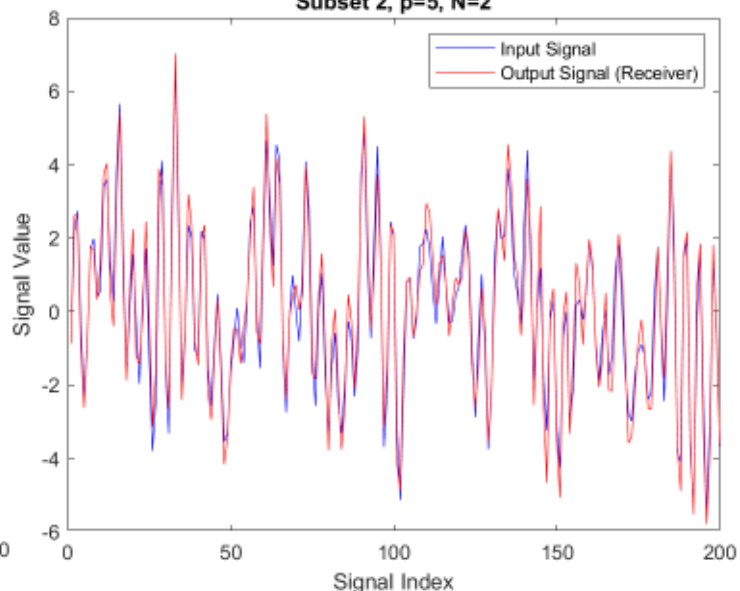
Τέλος, παρακάτω φαίνεται το αρχικό και το ανακατασκευασμένο σήμα στον δέκτη για καθένα από τα 4 υποσύνολα, και για τιμές του $p=5, 10$ και $N=1, 2, 3$:



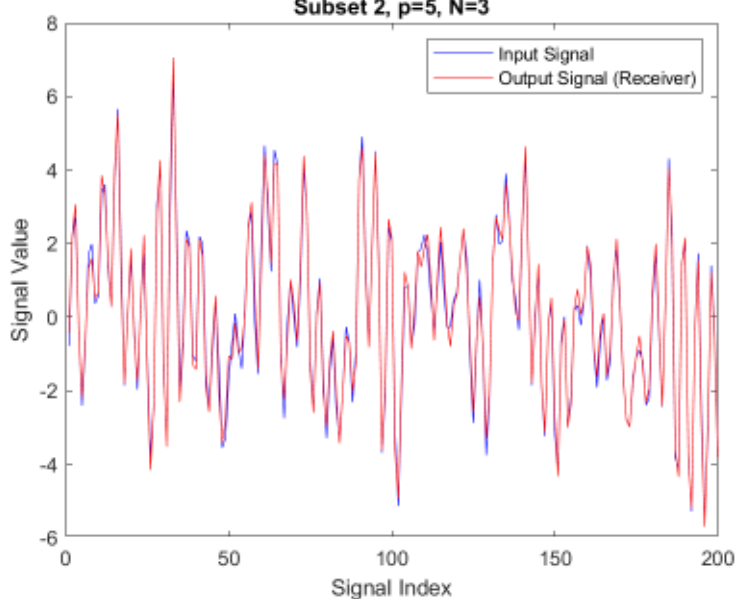
Subset 2, $p=5$, $N=1$



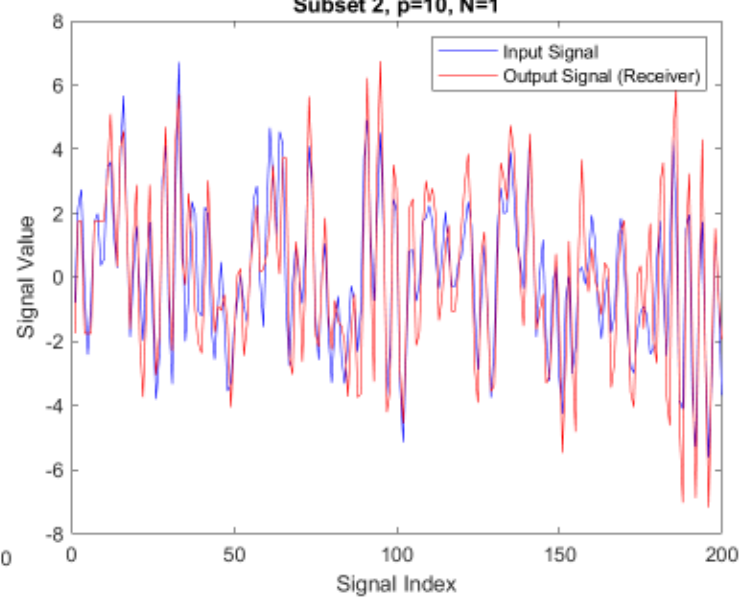
Subset 2, $p=5$, $N=2$



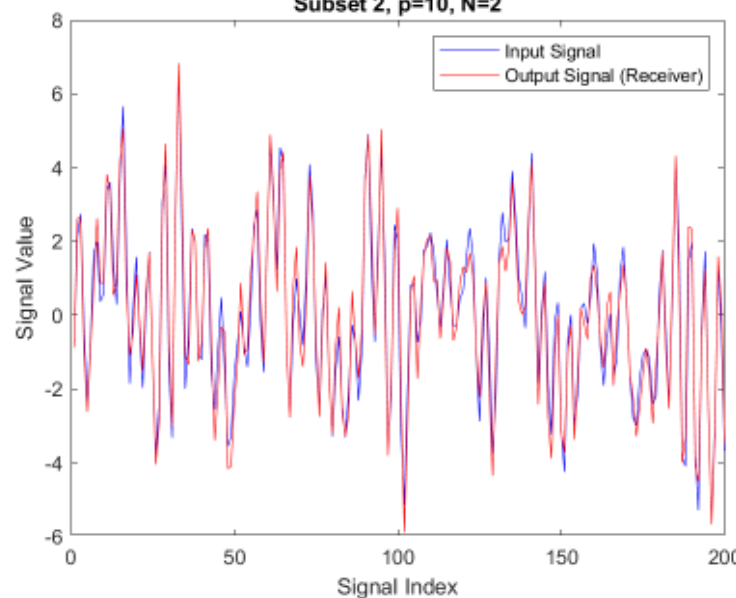
Subset 2, $p=5$, $N=3$



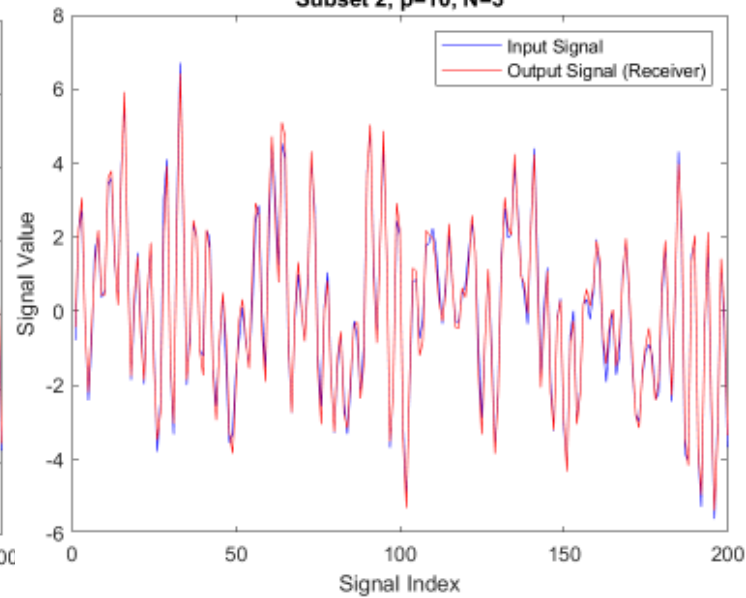
Subset 2, $p=10$, $N=1$



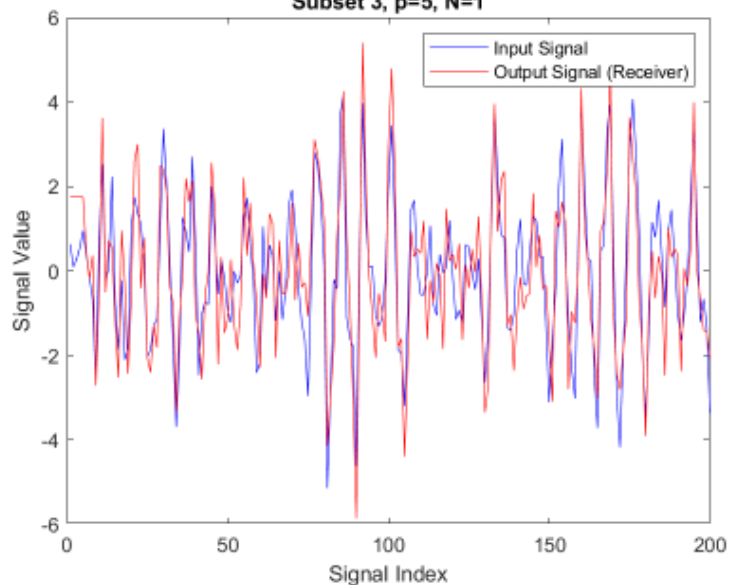
Subset 2, $p=10$, $N=2$



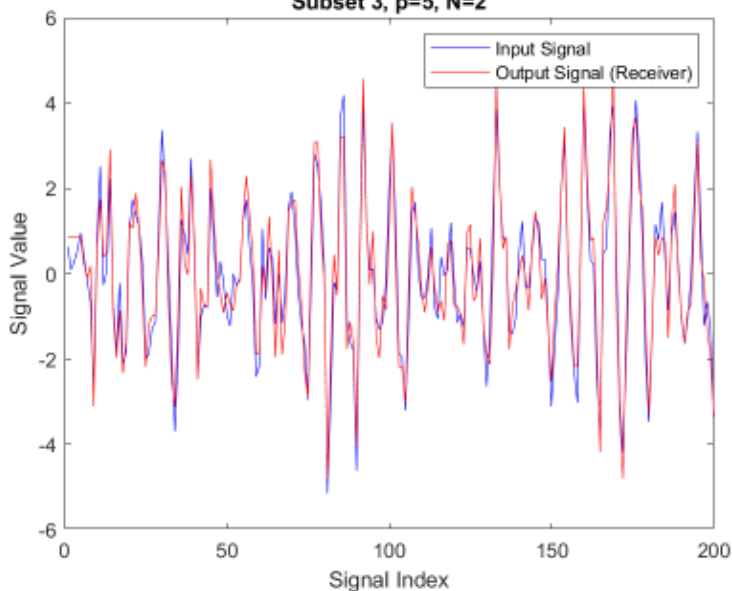
Subset 2, $p=10$, $N=3$



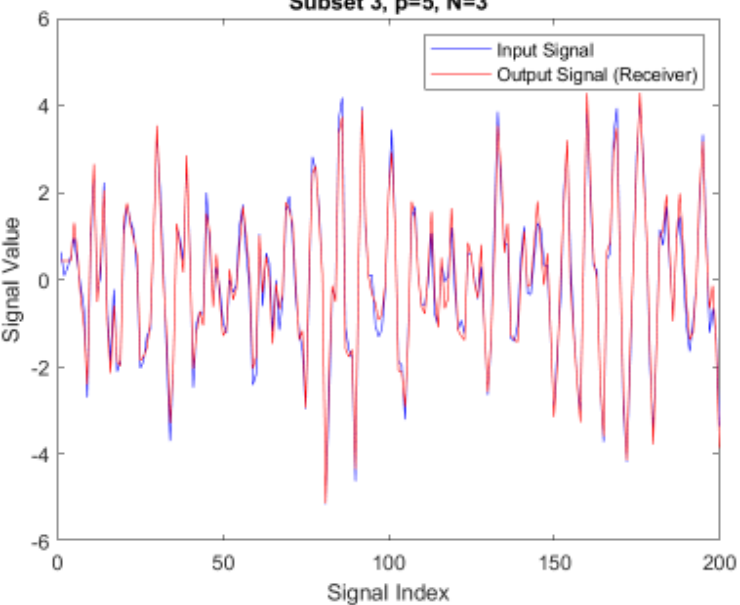
Subset 3, $p=5$, $N=1$



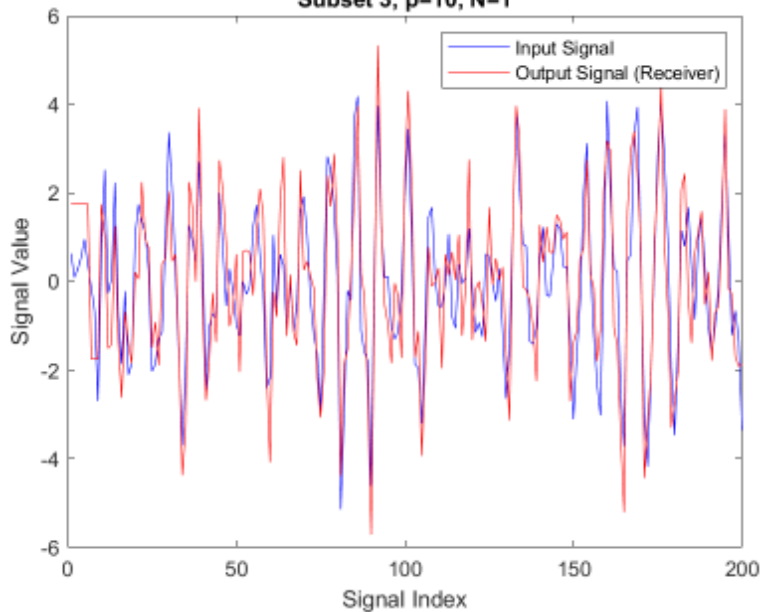
Subset 3, $p=5$, $N=2$



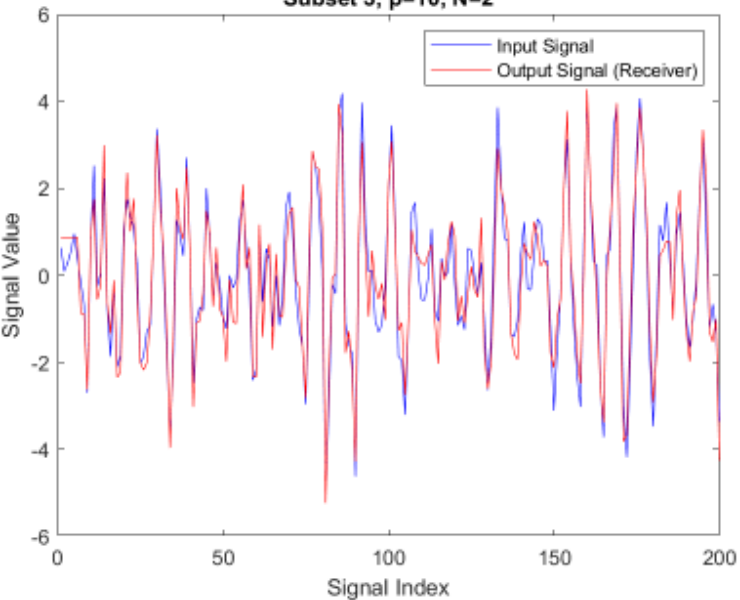
Subset 3, $p=5$, $N=3$



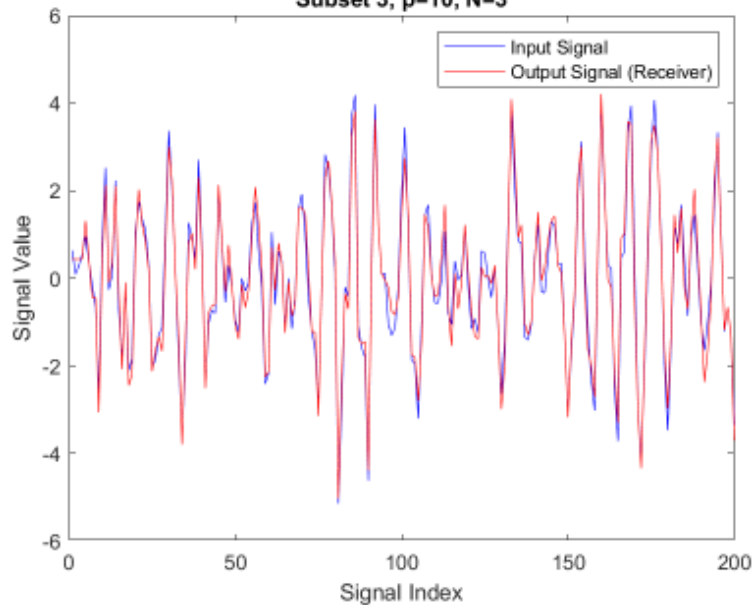
Subset 3, $p=10$, $N=1$

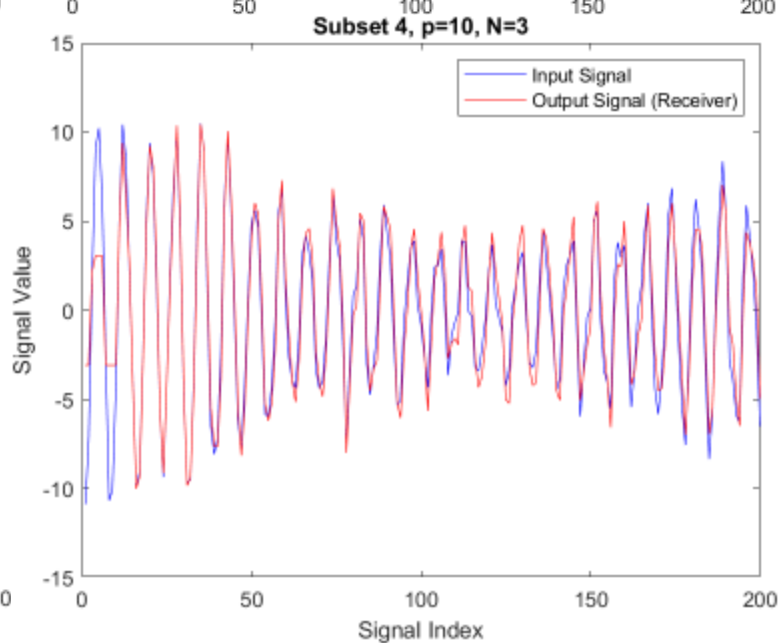
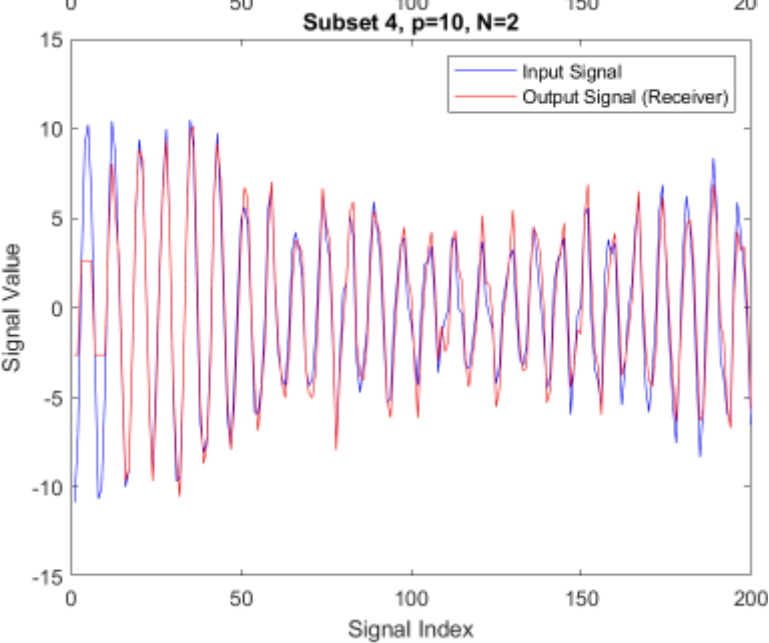
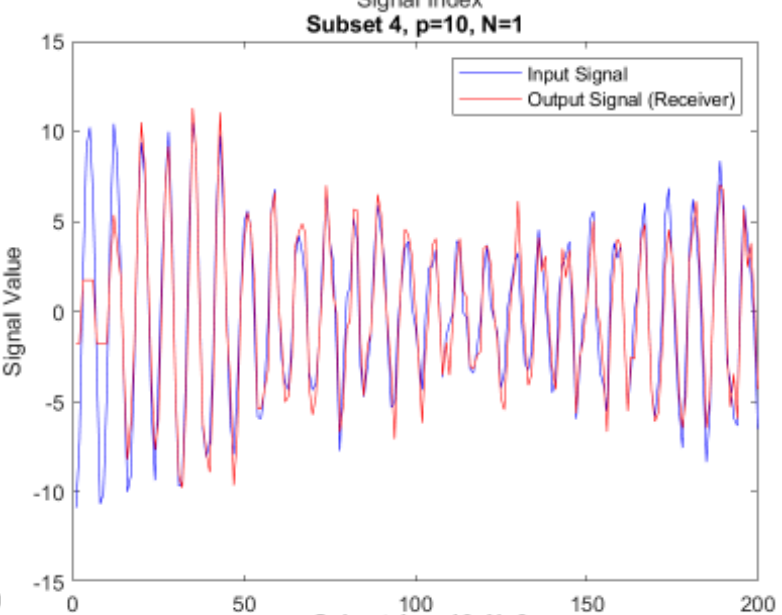
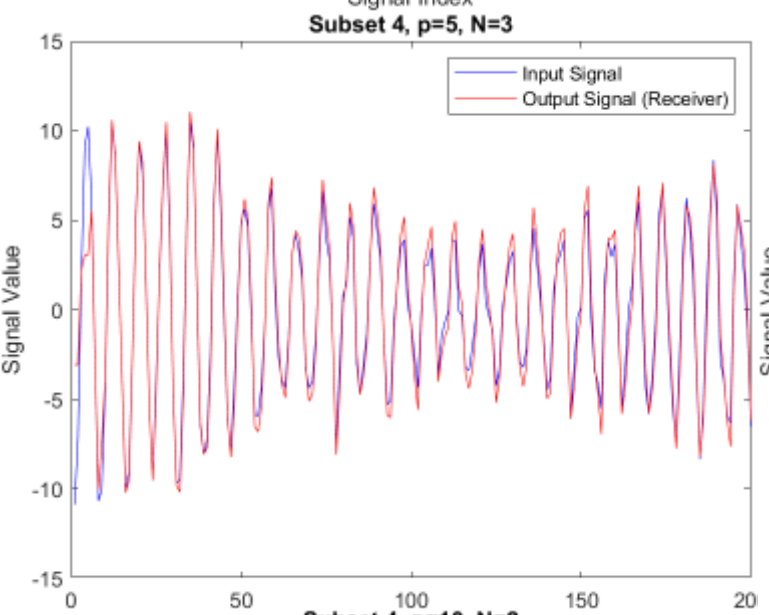
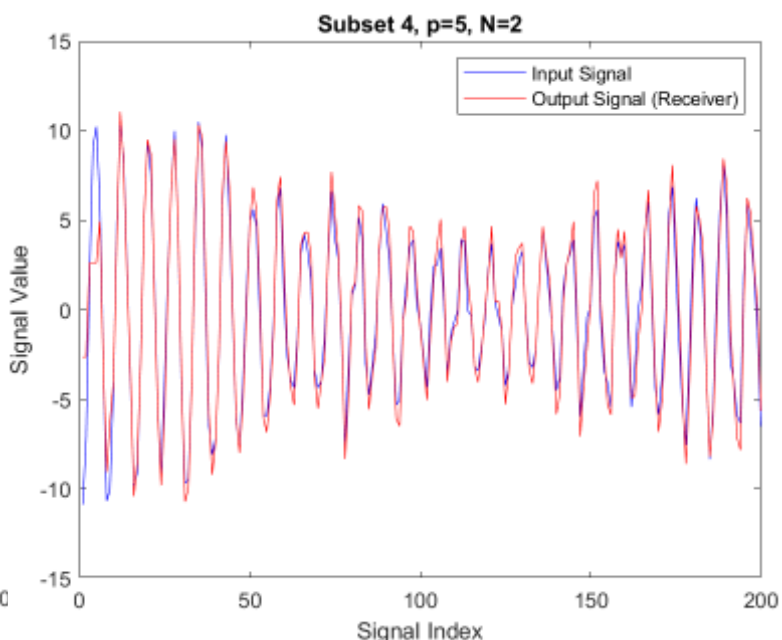
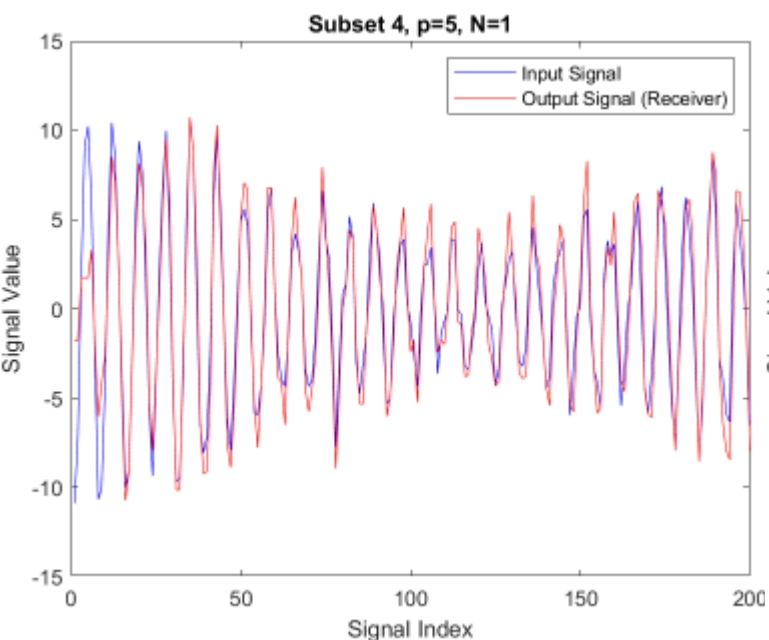


Subset 3, $p=10$, $N=2$



Subset 3, $p=10$, $N=3$





Παρατηρούμε ότι σε γενικές γραμμές οι αποδόσεις των υποσυνόλων είναι χειρότερες από τις αντίστοιχες του αρχικού μας σήματος. Πιο συγκεκριμένα, διακρίνουμε γενικά αυξημένο σφάλμα πρόβλεψης σε σχέση με το αρχικό σήμα, καθώς και σχετικά κακώς ανακατασκευασμένο σήμα, σε σχέση πάλι με το αρχικό μας, τόσο κατά την αύξηση του αριθμού συντελεστών πρόβλεψης, όσο και κατά την αύξηση του αριθμού bit κβάντισης. Ωστόσο το μόνο που έδειξε βελτίωση είναι το μέσο τετραγωνικό σφάλμα, το οποίο είναι αναμενόμενο καθώς έχουμε μικρότερο αριθμό τιμών. Η, κατά κύριο λόγο, χειρότερη απόδοση, μπορεί να οφείλεται σε αρκετούς λόγους, μερικοί από τους οποίους είναι οι εξής:

1. **Αρχικοποίηση Προβλεπτή:** Σε κάθε νέο υποσύνολο, ο προβλεπτής αρχικοποιείται ξανά, κάτι το οποίο είναι ανεπιθύμητο, καθώς εάν το σήμα εισόδου είναι, όπως το δικό μας, αρκετά προβλέψιμο, ο προβλεπτής δεν μπορεί να «μάθει», από προηγούμενες τιμές του σήματος σε άλλα υποσύνολα.
2. **Μείωση Απόδοσης Σημείων Έναρξης:** Λόγω της αρχικοποίησης του προβλεπτή σε κάθε νέο υποσύνολο, μπορεί να υπάρξει μείωση της απόδοσής του, λόγω έλλειψης επαρκών προηγούμενων δεδομένων για πρόβλεψη.
3. **Πιθανές Διαφορές στην Απόδοση της Πρόβλεψης:** Η απόδοση της πρόβλεψης μπορεί να διαφέρει μεταξύ των υποσυνόλων, ειδικά εάν κάθε υποσύνολο παρουσιάζει διαφορετικά χαρακτηριστικά σήματος. Αυτό μπορεί να επηρεάσει την συνολική απόδοση της μεθόδου στο σήμα.

Γενικότερα, αν και χωρίζοντας το σήμα σε μικρότερα υποσύνολα είναι ίσως πιο εύκολο στην αποθήκευση και την διαχείριση, για μηχανήματα με περιορισμένους υπολογιστικούς πόρους, εισάγονται σημαντικές διαταραχές στην συνέχεια και την ακρίβεια του σήματος και διαταράσσουν την ομαλή λειτουργία της εκάστοτε επεξεργασίας που προσπαθούμε να του επιβάλλουμε.

Κωδικας:

A Μέρος:

```
I = imread('parrot.png');
```

```
% Exercice 1a.
```

```
% Vector with all the distinct values of the matrix above.  
symbols1 = unique(I);
```

```
% Vector that counts the number of times each symbol occurs in the matrix  
% I.  
counts1 = histc(I(:), symbols1);
```

```
% To find the probability of appearance of each symbol, we must divide each  
% number in the counts vector with the number 3e4, which is the total  
% elements in the I matrix.  
% i-th value in symbols has the i-th symbol's prob of appearance.  
probabilities1 = counts1 ./ 3e4;
```

```
% Exercice 1b
```

```
% We construct the huffman dictionary for the symbols  
huffdict1 = huffmandict(symbols1, probabilities1);
```

```
% We calculate the entropy of the huffman dictionary
```

```

ent1 = entropy(probabilities1);

% We calculate the average length of the huffman code.
lbar1 = 0;
for i = 1:16
    lbar1 = lbar1 + (probabilities1(i) * length(huffdict1{i,2}));
end

% We calculate the efficiency of the huffman code, by dividing the entropy
% by the average length of the code.
eff1 = ent1 / lbar1;

% Exercise 2a.

l1 = size(I);
imageLength = l1(1)*l1(2);

% First we turn the matrix of the image into a row vector.
Ires = reshape(I',[],1);

% We brake the vector into pairs, resembling the pairs the expansion of the
% source would produce.
Iproc = [Ires(1:2:end),Ires(2:2:end)];

% Based on Iproc, we find the unique pairs that is the alphabet of the
% expanded source.
[symbols2,column_labels,row_labels] = unique(Iproc, "rows");

% We count how many times each pair is founded in the Iproc matrix and we
% calculate the probability of appearance for each symbol.
counts2 = histcounts(row_labels, max(row_labels));
totalCount = sum(counts2);
probabilities2 = counts2 ./ totalCount;
probabilities2 = probabilities2';

% We calculate the entropy of the expansion of the source
ent2 = 0;

for i = 1 : 174
    ent2 = ent2 - (probabilities2(i,1) * log2(probabilities2(i,1)));
end

% Reformatting the symbols2 matrix to a vector of strings.
symbols2form = cell(174,1);

for i = 1 : 174
    symbols2form{i} = [num2str(symbols2(i,1)), '-',
num2str(symbols2(i,2))];
end

% We construct the huffman dictionary for the symbols
huffdict2 = huffmandict(symbols2form,probabilities2);

% We calculate the average length of the huffman code.
lbar2 = 0;
for i = 1:174
    lbar2 = lbar2 + (probabilities2(i,1) * length(huffdict2{i,2}));
end

```

```

% We calculate the efficiency of the huffman code, by dividing the entropy
% by the average length of the code.
eff2 = ent2 / lbar2;

%Exercise 4.

% We encode the source using the huffman dictionary huff1 we found in the
% first exercise
encSrc = huffmanenco(Ires,huffdict1);

% We ensure the code works by decoding the encoded sequence using the same
% huffman dictionary.
decSrc = huffmandeco(encSrc, huffdict1);

% This is the binary depiction of the Image.
bI = reshape((dec2bin(typecast(I(:),'uint8'),4)-'0').',1,[]);

% We calculate the ratio between the bits used to encode the image using
% the huffman dictionary and the bits used to binarize the image.
binaryToHuffmanRatio = length(encSrc) / length(bI);

% Now we want to determine the possibility of correct transmission through
% the given channel by observing the vectors of input and output.

% First we pass the encoded vector through the channel.
output = binary_symmetric_channel(encSrc);

% We compare the two vectors to find where are the mistakes. The 1 in the
% comparison vector equals to a mistake.
comparison = xor(encSrc, output);
comparisonCounts = sum(comparison);

% We calculate the probability of the channel, by counting all the
% mistakes and dividing them with the length of the vector.
channelProb = comparisonCounts / length(encSrc);

% Because our channel is binary and symmetric, the capacity can be easily
% calculated as follows .
capacity = 1 + channelProb*log2(channelProb) + (1 - channelProb)*log2(1 -
channelProb);

% Binary entropy function H(p)
H_p = @(p) -p * log2(p) - (1 - p) * log2(1 - p);
H_pe = H_p(channelProb);

% Mutual Information I(X; Y) in a BSC is equal to the channel capacity
mutual_information = 1 - H_pe;

```

B Μέρος:

```

function [y_cap,centers] = my_quantizer(y,N,min_value,max_value)
% MY_QUANTIZER inputs:
% 1. y: the input signal to be quantized.
% 2. N: the number of bits used for the representation of the
%    representative values.
% 3. min_value: maximum value of quantization.

```

```

% 4. max_value: minimum value of quantization.
%
% MY_QUANTIZER outputs:
% 1. yn: the quantized signal.
% 2. centers: the representative values of each quantization level.

% We calculate the number of quantization levels (dynamic regions).
L = 2^N;

% We calculate the step size
D = (max_value - min_value) / L;

% We now have L levels of quantization each of size D. We now calculate the
% center of each quantization level as the middle of each level.

centers = min_value + D/2 : D : max_value - D/2;

% We preallocate the space for the quantized signal
y_cap = zeros(size(y));

% Quantize each element of the input signal. This is achieved by
% creating a vector with the absolute difference of the value with each of
% the centers. Then we find the position of the minimum value in that
% vector and we insert the value of the center in the same position in the
% quantized output signal.
for i = 1:length(y)
    [~, idx] = min(abs(centers - y(i)));
    y_cap(i) = centers(idx);
end
end

function [r, R, a] = prediction_filter(p, x)
% Prediction_filter inputs:
% 1. p: the number of previous values of the signal we want to keep.
% 2. x: the input signal.
%
% Prediction_filter outputs:
% 1. r: self-correlation vector of size px1.
% 2. R: self-correlation matrix of size p x p.
% 3. a: vector of the prediction filter coefficients.

N = length(x);
frac = 1/(N-p);
r = zeros(p,1);
R = zeros(p,p);
a = zeros(p,1);

% We calculate the rx(i) which go to r(i)
for i = 1 : p
    sum = 0;
    for n = p+1 : N
        sum = sum + (x(n)*x(n-i));
    end
    r(i) = sum*frac;
end

% We calculate the Rx(i - j) which go to the R(i,j)
for i = 1 : p
    for j = 1 : p

```

```

        sum = 0;
        for n = p+1 : N
            sum = sum + x(n - j)*x(n - i);
        end
        R(i,j) = sum*frac;
    end
end

% We calculate the coefficients, using the Yule Walker equations.
a = R\r;

end

function [predicted_signal] = predictor(a,n,y_cap_tone)
% predictor inputs:
% 1. a: The prediction coefficients used to predict the signal's values.
% 2. n: The size of the signal.
% 3. y_cap_tone: the processed output signal, which is the input of the
% predictor
%
% predictor outputs:
% 1. predicted_signal: the signal, whose values are the predictions of
% the input signal.

%preallocate the space of the vector
predicted_signal = zeros(n,1);

%first value is zero because the first iteration has no previous value of
%the input signal, to calculate a prediction
predicted_signal(1) = 0;

% i is equal to p+1 or length(a)+1, to ensure the i-j parameter inside the
% loop is positive. This means the first p values of the predicted_signal
% are 0. We could try to fill them out, though this would cause a additive
% error in the next calculations.
for i = length(a)+1 : n
    for j = 1 : length(a)
        predicted_signal(i) = predicted_signal(i) + a(j) * y_cap_tone(i-
j);
    end
end
end
end

function [y,y_cap_tone,output_signal,predicted_signal,a] =
DPCM_encoder(input_signal,p,N,min_value,max_value)
% DPCM_encoder inputs:
% 1. y: Prediction error.
% 2. y_cap_tone: processed output signal.
% 3. output_signal: the signal that gets transmitted to the receiver.
% 4. predicted_signal: the predicted signal.
% 5. a: the coefficients of the prediction of the signal.
%
% DPCM_encoder outputs:
% 1. input_signal: the signal that we want to transmit.
% 2. p: the number of coefficients we want.
% 3. N,min_value,max_value: See my_quantizer.m

% We initialize the vectors.
[~,~,a] = prediction_filter(p,input_signal);

```

```

predicted_signal = zeros(size(input_signal));
y = zeros(size(input_signal));
output_signal = zeros(size(input_signal));
y_cap_tone = zeros(size(input_signal));

% We implement the given functions of the exercise.
for i = 1: length(input_signal)
    y(i) = input_signal(i) - predicted_signal(i);
    [output_signal(i),~] = my_quantizer(y(i),N,min_value,max_value);
    y_cap_tone(i) = output_signal(i) + predicted_signal(i);
    predicted_signal = predictor(a, length(input_signal),y_cap_tone);
end
end
function [output_received_signal,predicted_received_signal] =
DPCM_decoder(received_signal,quantized_coeff)
% DPCM_decoder inputs:
% 1. received_signal: The signal that gets outputted at the transmitter
end.
% 2. quantized_coeff: The coefficients that get quantized at the
transmitter end and sent to the receiver beforehand..
%
% DPCM_decoder outputs:
% 1. output_received_signal: The signal that gets outputted at the
receiver end.
% 2. predicted_received_signal: The output of the predictor at the
receiver end.

output_received_signal = zeros(size(received_signal));
predicted_received_signal = zeros(size(received_signal));

for i = 1 : length(received_signal)
    output_received_signal(i) = received_signal(i) +
predicted_received_signal(i);
    predicted_received_signal = predictor(quantized_coeff,
length(received_signal),output_received_signal);
end
end

```

Παρακάτω παραθέτουμε κάποια scripts που χρησιμοποιήθηκαν για τα διαγράμματα, σε περίπτωση που μπορέσουν να βοηθήσουν στην διόρθωση και την κατανόηση του κώδικα παραπάνω:

Partition_plots:

```

% Script for partitioning the input_signal and based on the p,N values, it
% uses the DPCM_encoder and DPCM_decoder functions to calculate the
% output_signal in the receiver, for every subset.

```

```

%In case you want to view the input_signal with it's prediction error in
%the sender, replace the three commands with [y,~,~,~] =
DPCM_encoder(current_subset,p,N,-3.5,3.5);
%as well as the second plot with plot(y(1:200), 'r');

```

```

% we define the values of number of prediction coefficients and bits of
% quantization
p_values = [5, 10];
N_values = [1, 2, 3];

```

```

subset_size = 5000;

```

```

num_subsets = length(input_signal) / subset_size;

for subset = 1:num_subsets
    current_subset = input_signal((subset-1)*subset_size + 1 :
subset*subset_size);

    for p = p_values
        for N = N_values
            % for each combination of p,N we keep the signals we want to
            % see
            [~,~,output_signal,~,a] = DPCM_encoder(current_subset,p,N,-
3.5,3.5);
            [quant_coef,~] = my_quantizer(a,8,-2,2);
            [output_received_signal,~] =
DPCM_decoder(output_signal,quant_coef);

            %We keep the first 200 samples of the signals, in order to see
            %them more clearly
            figure;
            plot(current_subset(1:200), 'b');
            hold on;
            plot(output_received_signal(1:200), 'r');

            title(['Subset ', num2str(subset), ', p=', num2str(p), ', N=',
num2str(N)]);
            xlabel('Signal Index');
            ylabel('Signal Value');
            legend('Input Signal', 'Output Signal (Receiver)');
        end
    end
end

MSE:
% We calculate the Mean Square Error for every value of p, N, and putting
% the values of a(prediction coefficients) in a cell to plot them easier.
N_values = 1:3;
p_values = 5:10;
mse_results = zeros(length(p_values), length(N_values));
prediction_coefficients = cell(length(p_values), length(N_values));

for i = 1:length(p_values)
    p = p_values(i);
    for j = 1:length(N_values)
        N = N_values(j);

        [~,~,~,predicted_signal,a] = DPCM_encoder(input_signal,p,N,-
3.5,3.5);

        %MSE
        mse_results(i, j) = immse(input_signal, predicted_signal);
        prediction_coefficients{i, j} = a;
    end
end

% graph
figure;
plot(N_values, mse_results', '-o');
title('MSE ως προς το N για διάφορες τιμές του p');

```

```

xlabel('N (bits κβαντισμού)');
ylabel('MSE');
legend(arrayfun(@(p) sprintf('p = %d', p), p_values, 'UniformOutput',
false));
grid on;

MSE_partitions:
N_values = 1:3;
p_values = 5:10;

subset_size = 5000;

num_subsets = length(input_signal) / subset_size;

mse_results = zeros(length(p_values), length(N_values), num_subsets);
prediction_coefficients = cell(length(p_values), length(N_values),
num_subsets);

for subset = 1:num_subsets
    current_subset = input_signal((subset-1)*subset_size + 1 :
subset*subset_size);

    for i = 1:length(p_values)
        p = p_values(i);
        for j = 1:length(N_values)
            N = N_values(j);

            [~, ~, ~, predicted_signal, a] = DPCM_encoder(current_subset,
p, N, -3.5, 3.5);

            % MSE
            mse_results(i, j, subset) = immse(current_subset,
predicted_signal);
            prediction_coefficients{i, j, subset} = a;
        end
    end

    figure;
    plot(N_values, squeeze(mse_results(:, :, subset))', '-o');
    title(['MSE ως προς το N για διάφορες τιμές του p, Υποσύνολο ',
num2str(subset)]);
    xlabel('N (bits κβαντισμού)');
    ylabel('MSE');
    legend(arrayfun(@(p) sprintf('p = %d', p), p_values, 'UniformOutput',
false));
    grid on;
end

```