



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Ηλεκτρολόγων Μηχανικών και
Μηχανικών Υπολογιστών

Εισαγωγή στις Τηλεπικοινωνίες

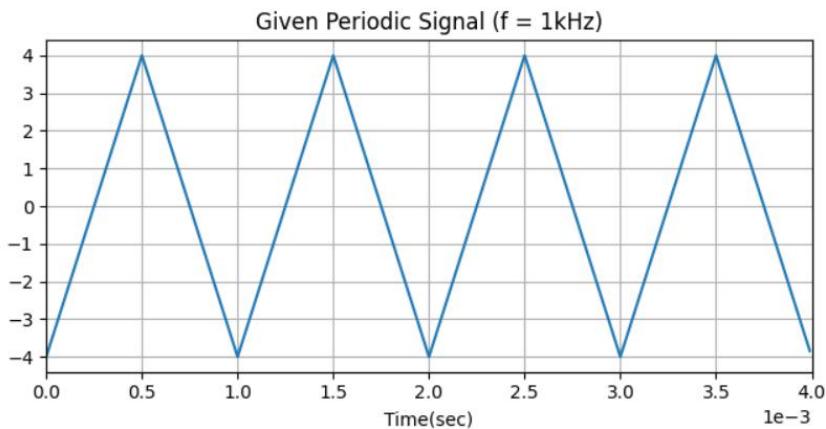
Εργαστηριακή Άσκηση, Ακαδ. Έτος 2021-22

Απαντήσεις στις Ασκήσεις

Ραπτόπουλος Πέτρος
Ομάδα 11 (1 άτομο)

1^ο Ερώτημα:

Έχουμε την παρακάτω τριγωνική περιοδική παλμοσειρά πλάτους $A = 4V$ και συχνότητας $f_m = 1kHz$:



```

7 def tri(t): # define triangular function for each time point
8     t = t % 0.001 # find the mod in order to apply
9         # formulas for the first pulse
10    if 0 <= t and t <= 0.0005: # 0 <= t <= T/2
11        return 16000*t-4
12    elif 0.0005 <= t and t <= 0.001: # T/2 <= t <= T
13        return -16000*t+12
14
15 vtri = np.vectorize(tri)
16 # make the tri() function an element-wise one

```

```

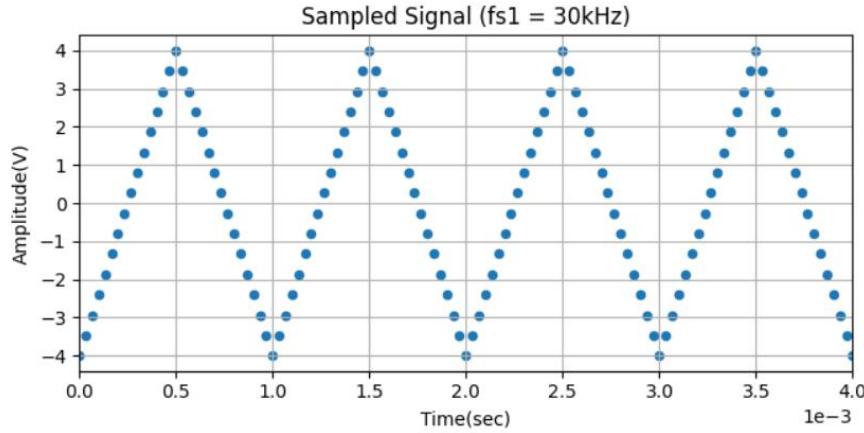
18 fm = 1000 # f = 1kHz
19 T = 0.001 # T = 10^-3 sec
20 conStep = 0.00001 # Step small enough to consider
21             # the outcome continuous
22 t = np.arange(0,4*T, conStep) # define time axis
23 # span the time values over 4 periods
24 # our signal is continuous despite the discrete
25 # representation
26
27 y = vtri(t) # y_i = tri(t_i)
28
29 fig1 = plt.figure(1) # create a new figure
30 ax = fig1.add_subplot(221) # create one subplot
31 plt.xlabel("Time(sec)")
32 plt.ylabel("Amplitude(V)")
33 plt.title("Given Periodic Signal (f = 1kHz)")
34 plt.grid() # use the grid
35 ax.plot(t,y)
36 ax.set_xlim(left = 0, right = 4*T) # start from 0 and end
37             # just before 4*T
38
39 plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
40 # use the scientific (exponential) notation for the x axis

```

Κώδικας σε Python

Η υλοποίηση των κώδικα της τριγωνικής περιοδικής συνάρτησης φαίνεται στα αριστερά.

(a'.i): Δειγματοληπτούμε το παραπάνω σήμα με συχνότητα $f_{s1} = 30f_m = 30kHz$, δηλαδή κάθε $T_{s1} \approx 0.03ms$ λαμβάνουμε δείγμα. Έχουμε τη παρακάτω γραφική παράσταση για τα δείγματα μετά τη δειγματοληψία.



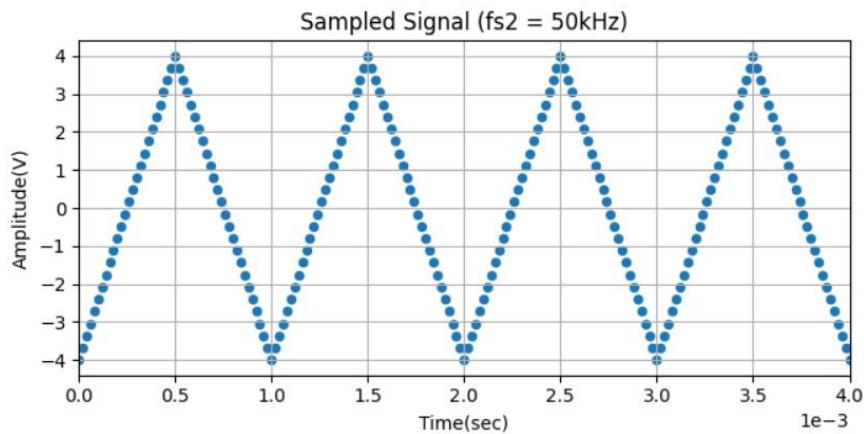
```

46 fs1 = 30*fm
47 Ts1 = 1/fs1
48 t1 = np.arange(0,4*T+Ts1, Ts1) # define time axis
49 # the instruction np.arange() isn't end-inclusive
50 # we must add Ts1 to the end in order to include
51 # the sample value at t = 4*T
52
53 y1 = vtri(t1) # y_i = tri(t_i), for all i
54
55 ax1 = fig1.add_subplot(222) # create one subplot
56 plt.xlabel("Time(sec)")
57 plt.ylabel("Amplitude(V)")
58 plt.title("Sampled Signal (fs1 = 30kHz)")
59 plt.grid() # use the grid
60 ax1.scatter(t1,y1, s=20, marker='o') # adjust size and marker
61 ax1.set_xlim(left = 0, right = 4*T) # start from 0 and end
62             # just before 4*T
63
64 plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
65 # use the scientific (exponential) notation for the x axis

```

Κώδικας σε Python

(a'.ii): Δειγματοληπτούμε το παραπάνω σήμα με συχνότητα $f_{s2} = 50f_m = 50kHz$, δηλαδή κάθε $T_{s2} \approx 0.02ms$ λαμβάνουμε δείγμα. Έχουμε τη παρακάτω γραφική παράσταση για τα δείγματα μετά τη δειγματοληψία:

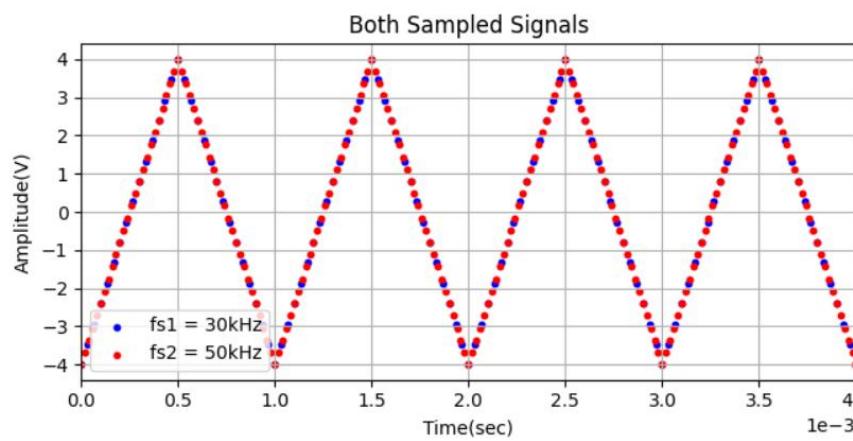


```

70 fs2 = 50*fm
71 Ts2 = 1/fs2
72 t2 = np.arange(0,4*T+Ts2, Ts2) # define time axis
73 # the instruction np.arange() isn't end-inclusive
74 # we must add Ts2 to the end in order to include
75 # the sample value at t = 4*T
76
77 y2 = vtri(t2) # y_i = tri(t_i), for all i
78
79 ax2 = fig1.add_subplot(223) # create one subplot
80 plt.xlabel("Time(sec)")
81 plt.ylabel("Amplitude(V)")
82 plt.title("Sampled Signal (fs2 = 50kHz)")
83 plt.grid() # use the grid
84 ax2.scatter(t2,y2, s=20, marker='o') # adjust size and marker
85 ax2.set_xlim(left = 0, right = 4*T) # start from 0 and end
86             # just before 4*T
87
88 plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
89 # use the scientific (exponential) notation for the x axis

```

(a'.iii): Έχουμε τη παρακάτω κοινή γραφική παράσταση για τα δείγματα και των δύο δειγματοληψιών:



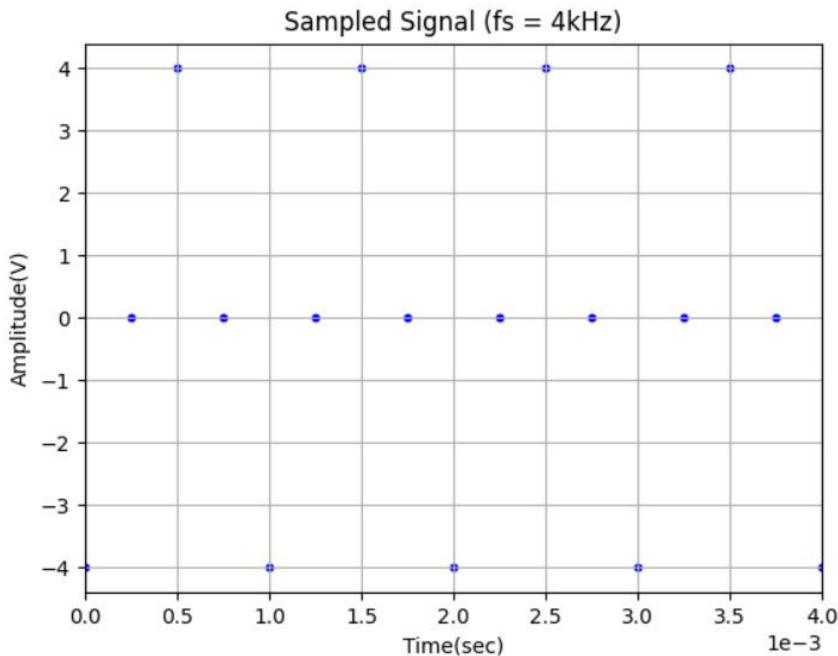
```

93 ax3 = fig1.add_subplot(224) # create one subplot
94 plt.xlabel("Time(sec)")
95 plt.ylabel("Amplitude(V)")
96 plt.title("Both Sampled Signals")
97 plt.grid() # use the grid
98 ax3.scatter(t1,y1, s=10, c='b', marker='o', label='fs1 = 30kHz')
99 # adjust size, marker, color and label
100 ax3.scatter(t2,y2, s=10, c='r', marker='o', label='fs2 = 50kHz')
101 # adjust size, marker, color and label
102 ax3.set_xlim(left = 0, right = 4*T) # start from 0 and end
103 # just before 4*T
104 ax3.legend() # show the Legends on the plot
105 plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
106 # use the scientific (exponential) notation for the x axis
107
108 fig1.tight_layout() # adjust the spacing between subplots in
109 # order to avoid text overlapping

```

Κώδικας σε Python

(b'): Δειγματοληπτούμε το δοθέν σήμα με συχνότητα $f_s = 4f_m = 4\text{kHz}$, δηλαδή κάθε $T_{s1} \approx 0.25\text{ms}$ λαμβάνουμε δείγμα. Έχουμε τη παρακάτω γραφική παράσταση για τα δείγματα μετά τη δειγματοληψία:



```

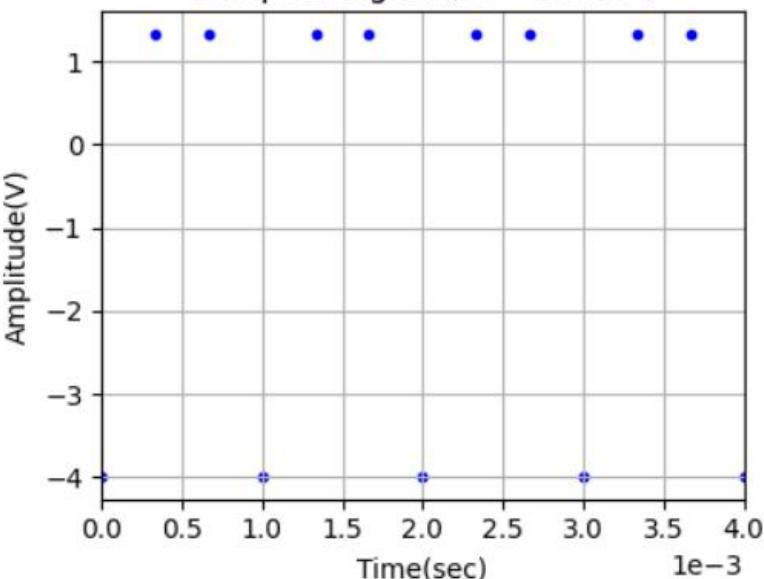
114 fs = 4*f_m
115 Ts = 1/fs
116 t = np.arange(0,4*T+Ts, Ts) # define time axis
117 # the instruction np.arange() isn't end-inclusive
118 # we must add Ts1 to the end in order to include
119 # the sample value at t = 4*T
120
121 y = vtri(t) # y_i = tri(t_i)
122
123 fig2 = plt.figure(2) # create a new figure
124 ax = fig2.add_subplot(111) # create one subplot
125 plt.xlabel("Time(sec)")
126 plt.ylabel("Amplitude(V)")
127 plt.title("Sampled Signal (fs = 4kHz)")
128 plt.grid() # use the grid
129 ax.scatter(t,y, s=10, c='b', marker='o')
130 # adjust size, marker, color and label
131 ax.set_xlim(left = 0, right = 4*T) # start from 0 and end
132 # just before 4*T
133
134 plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
135 # use the scientific (exponential) notation for the x axis
136
137 fig1.tight_layout() # adjust the spacing between subplots in
138 # order to avoid text overlapping

```

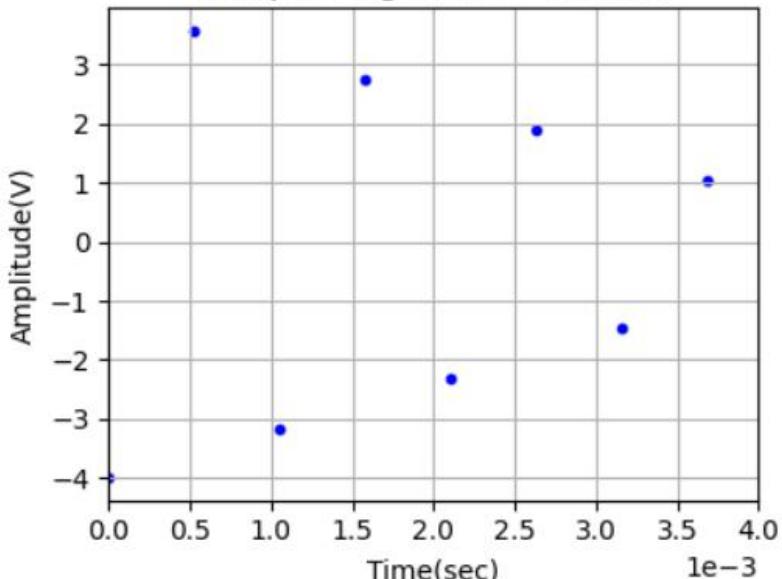
Κώδικας σε Python

Παρατηρούμε ότι η δειγματοληψία περιέχει τα σημεία μηδενισμού και τα ακρότατα της συνάρτησης $y(t)$. Προσαρμόζουμε τον παραπάνω κώδικα ώστε να απεικονίσουμε δειγματοληψίες διαφορετικών συχνοτήτων.

Sampled Signal (fs = 3.0kHz)



Sampled Signal (fs = 1.9kHz)



Έχουμε το διθέν τριγωνικό σήμα $y(t)$. Μοντελοποιούμε το αποτέλεσμα της δειγματοληψίας ως τον πολλαπλασιασμό του σήματος συνεχούς χρόνου με την κρονοστική παλμοσειρά. Για να βρούμε το φάσμα του προκύπτοντος σήματος, εφαρμόζουμε τον μετασχηματισμό Fourier στον προαναφερθέν πολλαπλασιασμό. Κατά αυτόν τον τρόπο, καταλήγουμε στην συνέλιξη των μετασχηματισμών Fourier, που μας υποδεικνύει την μετατόπιση του φάσματος (του σήματος συνεχούς χρόνου) κατά ακέραια πολλαπλάσια της συχνότητας δειγματοληψίας. Δηλαδή έχουμε για σήμα $x(t) = y(t)$:

$$X_s(f) = F\{x_s(t)\} = F\{x(t) \cdot s(t)\} = F\{x(t)\} * F\{s(t)\} = X(f) * S(f) = X(f) * \left[\frac{1}{T_s} \sum_{m=-\infty}^{m=+\infty} \delta(f - m \cdot f_s) \right]$$

$$\text{Άρα } X_s(f) = \frac{1}{T_s} \sum_{m=-\infty}^{m=+\infty} X(f - m \cdot f_s)$$

Παρατηρούμε ότι για να μην υπάρχει επικάλυψη μεταξύ των διαδοχικών επαναλήψεων του αρχικού φάσματος $X(f)$ γύρω από τις “κεντρικές” συχνότητες $m \cdot f_s$ πρέπει η ελάχιστη απόσταση μεταξύ των κεντρικών συχνοτήτων να είναι μεγαλύτερη από το φασματικό εύρος $2B$, δηλαδή πρέπει να ισχύει $f_s > 2B$. Διαφορετικά θα έχουμε επικάλυψη μεταξύ των διαδοχικών επαναλήψεων του φάσματος $X(f)$ καθιστώντας μη δυνατή την εξαγωγή του αρχικού φάσματος $X(f)$ από το προκύπτον φάσμα $X_s(f)$. Συνεπώς το σήμα $x(t)$ δεν μπορεί να ανακατασκευαστεί από το σήμα $x_s(t)$. Έχουμε δηλαδή αναδίπλωση (aliasing).

Για την ανακατασκευή του σήματος $x(t)$ από τα δείγματα της $x_s(t)$, εφαρμόζουμε στην τελευταία ιδανικό βαθυπερατό φίλτρο με συχνότητα αποκοπής $f_c = f_s/2$. Δηλαδή έχουμε την απόκριση συχνότητας:

$$H(f) \begin{cases} T_s, |f| \leq \frac{f_s}{2} \\ 0, |f| > \frac{f_s}{2} \end{cases}$$

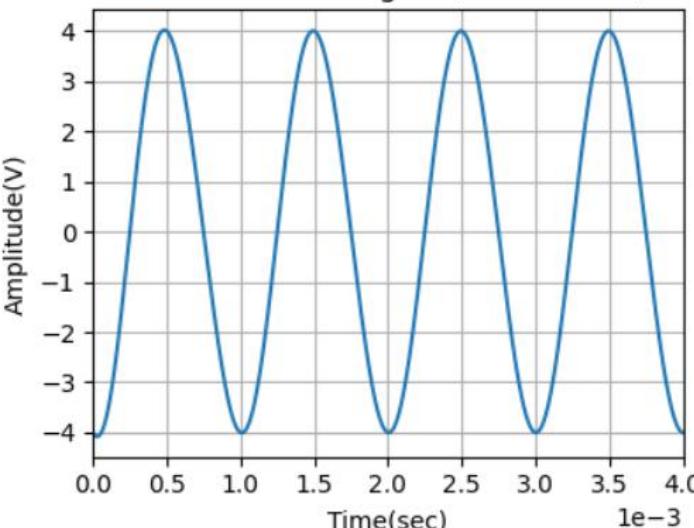
Άρα έχουμε την κρονοστική απόκριση, $h(t) = F^{-1}\{H(f)\} = \text{sinc}(\frac{t}{T_s})$. Συνεπώς έχουμε την έξοδο του φίλτρου:

$$y(t) = x(t) = x_s(t) * h(t) = \left[\sum_{n=-\infty}^{n=+\infty} x(nT_s) \cdot \delta(t - n \cdot T_s) \right] * \text{sinc}(\frac{t}{T_s}) = \sum_{n=-\infty}^{n=+\infty} x(nT_s) \cdot \text{sinc}(\frac{t - n \cdot T_s}{T_s})$$

Μόλις αποδείξαμε το Θεώρημα Δειγματοληψίας, έχοντας υποθέσει φασματικά περιορισμένο σήμα $x(t)$, με μέγιστη φασματική συνιστώσα στη συχνότητα B .

Σύμφωνα με τα παραπάνω, προσπαθούμε να ανακατασκευάσουμε το αρχικό σήμα, από τα δείγματά του με συχνότητα δειγματοληψίας $f_s = 4\text{kHz}$.

Reconstructed Signal ($f_s = 4.0\text{kHz}$)



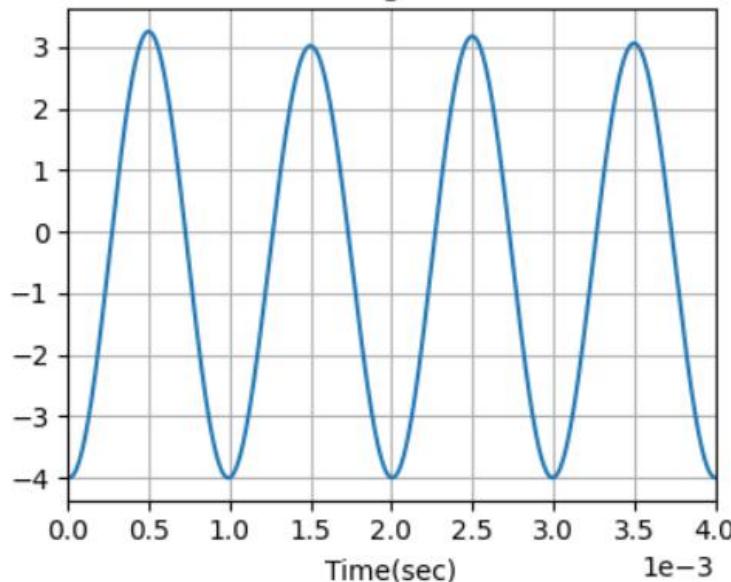
```
def sinc(x): # define sinc function for each time point
    if x == 0:
        return 1
    else:
        return math.sin(pi*x)/(pi*x)

# Try to reconstruct the signal from its samples
yrecon = np.empty(t.size)
for i in range(t.size):
    yrecon[i] = 0
    for n in range(ys.size):
        yrecon[i] += ys[n]*sinc((i*conStep-n*Ts)/Ts)
# From the theory we have y(t)=sum(y[nTs]*sinc((t-nTs)/Ts))
# for each n (integer) from -00 to +00
```

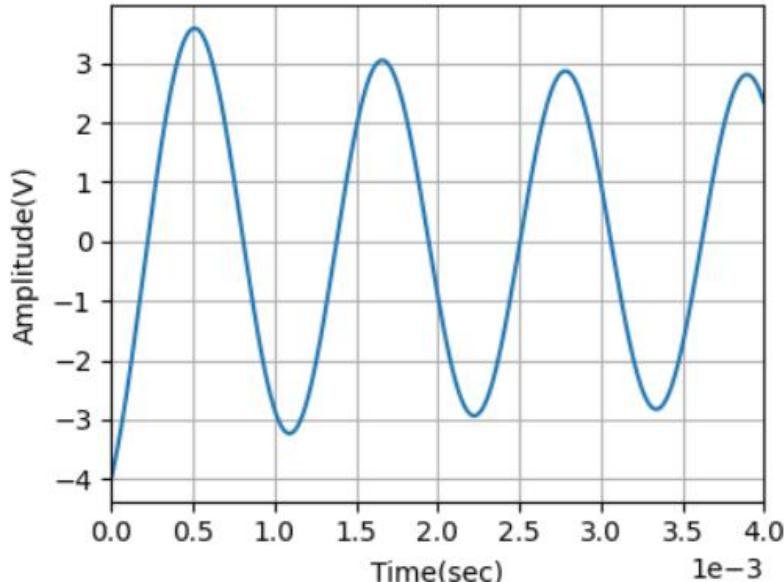
Κώδικας σε Python

Όπως φαίνεται από το παραπάνω διάγραμμα, παρόλο που τα δείγματα με $f_s = 4f_m$ είναι πολύ λιγότερα από αυτά των δειγματοληψιών με $f_s = 30f_m$ / $f_s = 30f_m$ μπορούμε να ανακατασκευάσουμε το δοθέν σήμα. Σύμφωνα με το Θεώρημα Δειγματοληψίας η ελάχιστη θεωρητική f_s ώστε να είναι δυνατή η ακριβής ανακατασκευή του σήματος υπό συζήτηση είναι $f_{s(\min)} = 2f_m = 2\text{kHz}$. (η ισότητα δεν ισχύει πάντα) Προσεγγίζοντας το θέμα πρακτικά, δοκιμάζουμε διαφορετικές συχνότητες δειγματοληψίας και προσπαθούμε να ανακατασκευάσουμε το αρχικό σήμα. Το φάσμα του τριγωνικού σήματος δεν είναι βαθυπερατό, περιέχει άπειρες αρμονικές συχνοτικές συνιστώσες. (Ανάπτυγμα Fourier: $y(t) = \sum_{n=1}^{n=+\infty} \frac{-32}{(2n-1)^2\pi^2} \cdot \cos(2n-1)\omega t$) Η επιλογή του φασματικού εύρους B γίνεται αυθαίρετα ανάλογα με την ακρίβεια που θέλουμε να πετύχουμε. Το σήμα παραμορφώνεται, αφού φίλτραρεται μέρος του συχνοτικού του περιεχομένου. Για $f_B = f_m$ (το τριγωνικό σήμα προσεγγίζεται με 81% ακρίβεια από ημιτονοειδή συνάρτηση): Παρατηρούμε ότι για συχνότητες $f_s \geq 2f_m$ (π.χ. για $f_s = 3\text{kHz}$) το σήμα ανακατασκευάζεται επιτυχώς. Ωστόσο παρατηρούνται μικρές παραμορφώσεις που αφορούν το πλάτος του, λόγω της προσέγγισης του άπειρου αθροίσματος (του θεωρήματος δειγματοληψίας) με το αντίστοιχο πεπερασμένο. Επίσης, και για $f_s < 2f_m$ παρατηρούμε ότι το ανακατασκευασμένο σήμα έχει μικρότερη συχνότητα από αυτή του αρχικού σήματος, έχουμε δηλαδή εμφανής παραμόρφωση του φάσματος. Συνεπώς, και υπό πρακτική προσέγγιση, η ελάχιστη συχνότητα ανακατασκευής σήματος είναι $f_{s(\min)} = 2f_B$. Στο ίδιο συμπέρασμα θα καταλήγαμε για οποιαδήποτε επιλογή του εύρους B . Για $f_B = 3f_m$ έχουμε 90% ακρίβεια για το τριγωνικό σήμα, ενώ για $f_B = 5f_m$ έχουμε 93% ακρίβεια.

Reconstructed Signal ($f_s = 3.0\text{kHz}$)



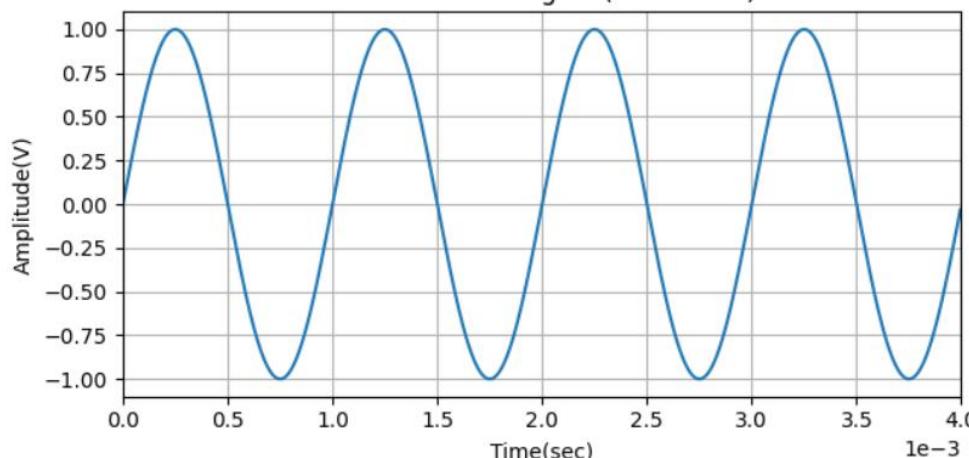
Reconstructed Signal ($f_s = 1.9\text{kHz}$)

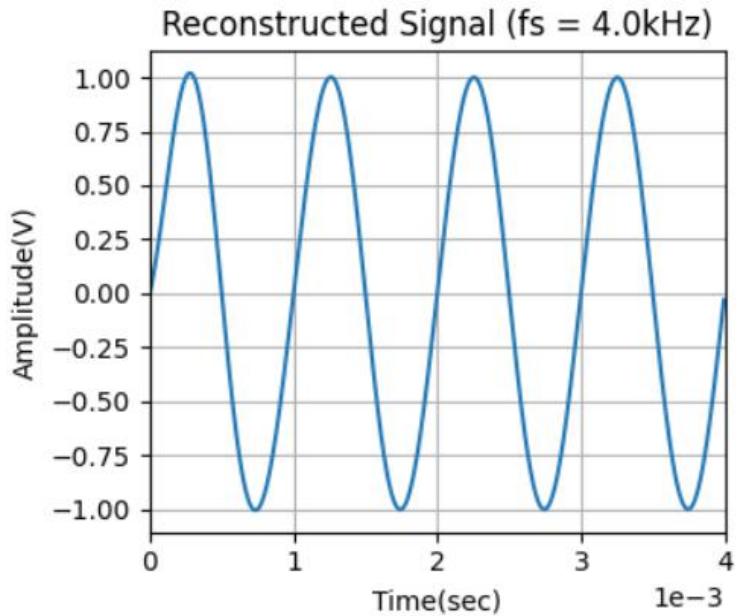
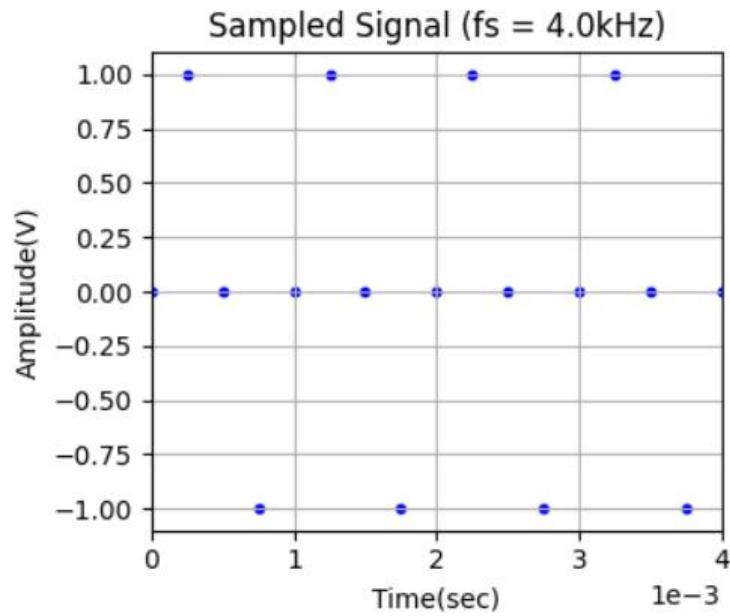
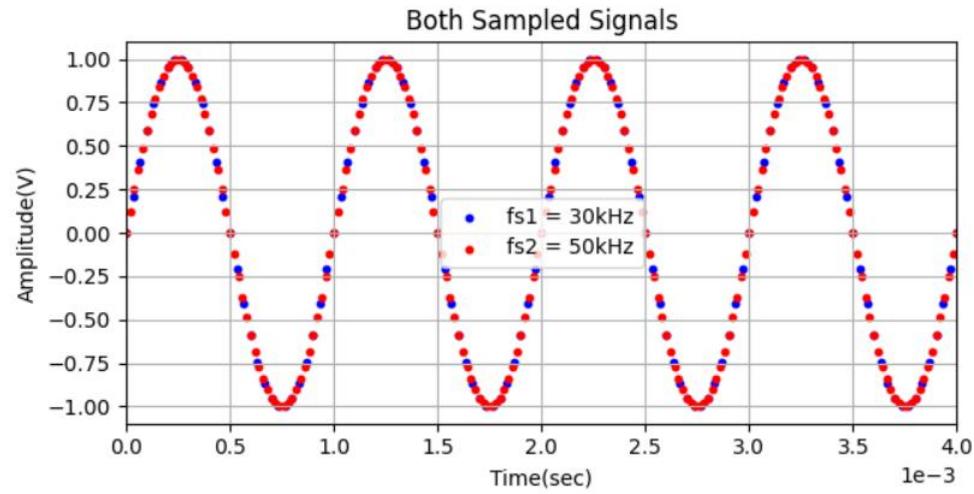
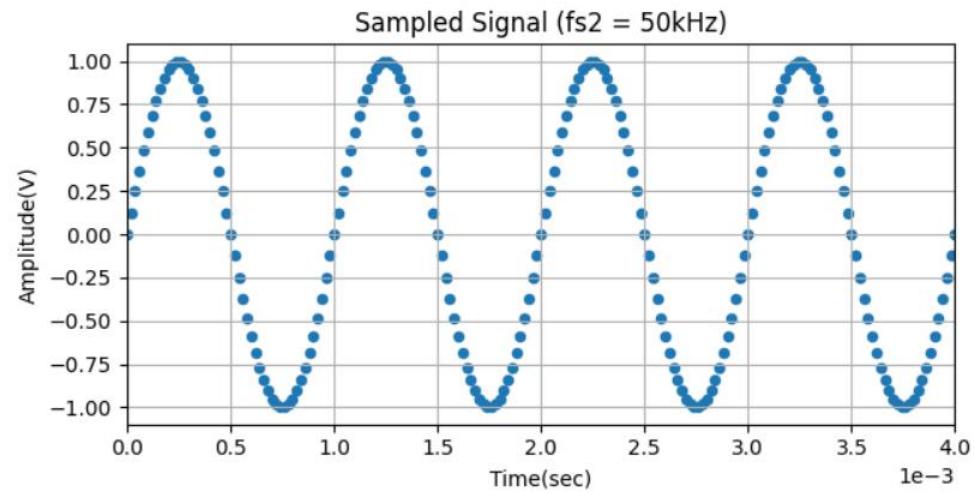
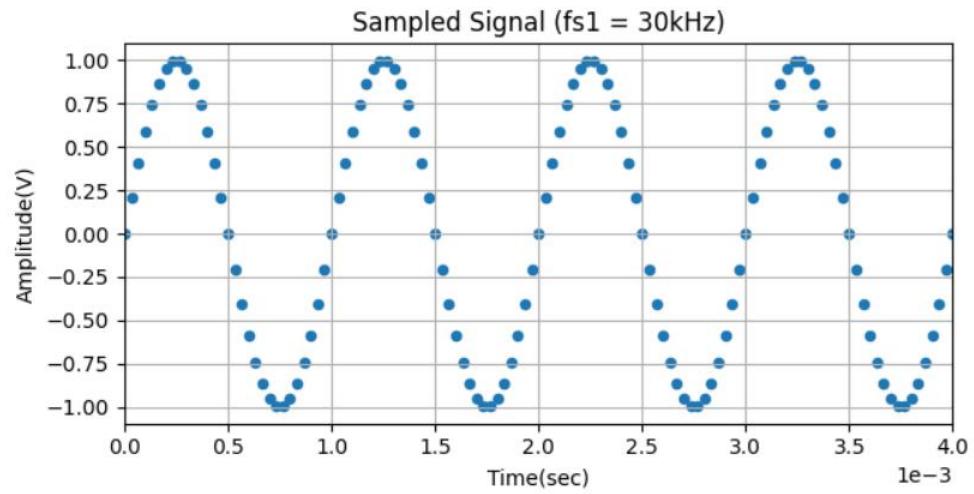


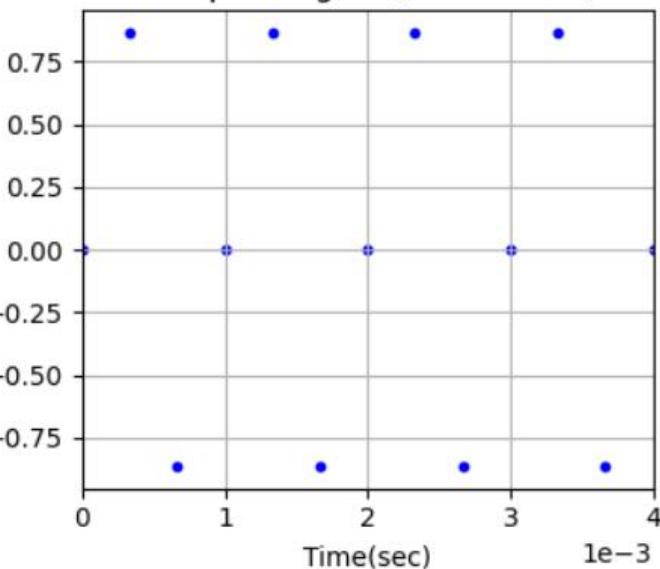
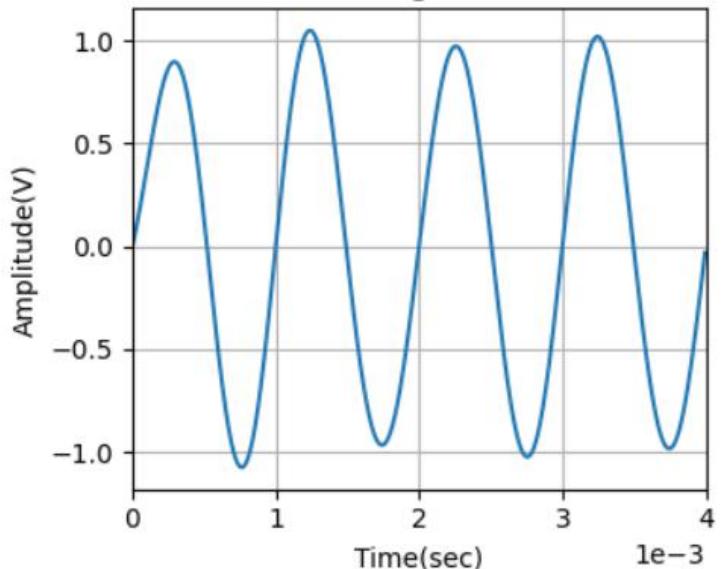
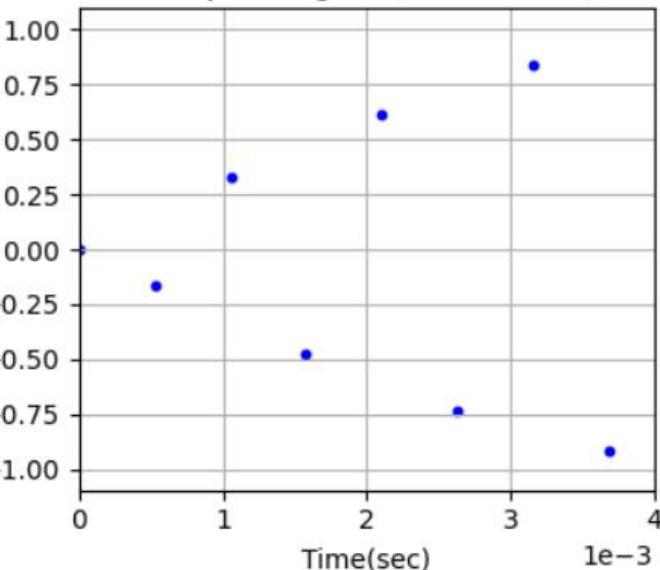
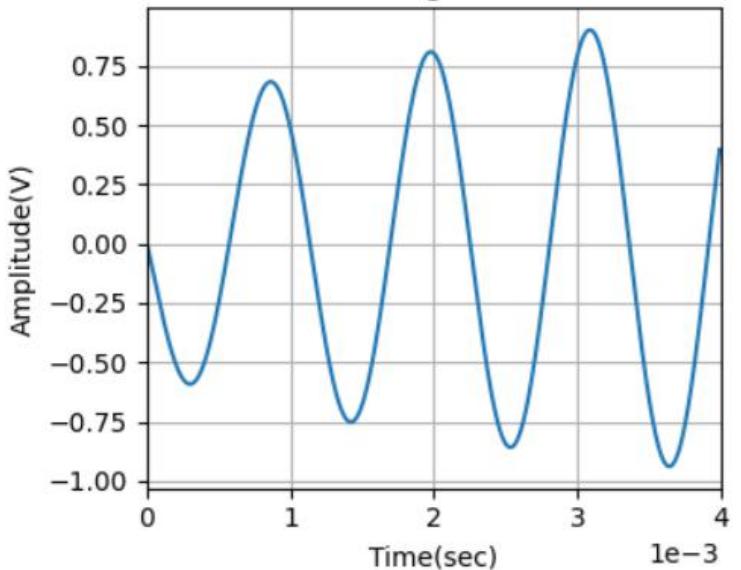
(c'): Έχουμε $z(t) = \sin(2\pi \cdot 1000t)$. Επαναλαμβάνουμε τα ερωτήματα α', β' για το σήμα $z(t)$. Για να αποφύγουμε περιττές επαναλήψεις του κώδικα, παραμετροποιούμε κατάλληλα τον κώδικα των προηγούμενων ερωτημάτων έτσι ώστε να εφαρμόζει κατάλληλα σε όλες τις ζητούμενες περιπτώσεις. Άρα έχουμε:

(i):

Given Periodic Signal ($f_m = 1\text{kHz}$)





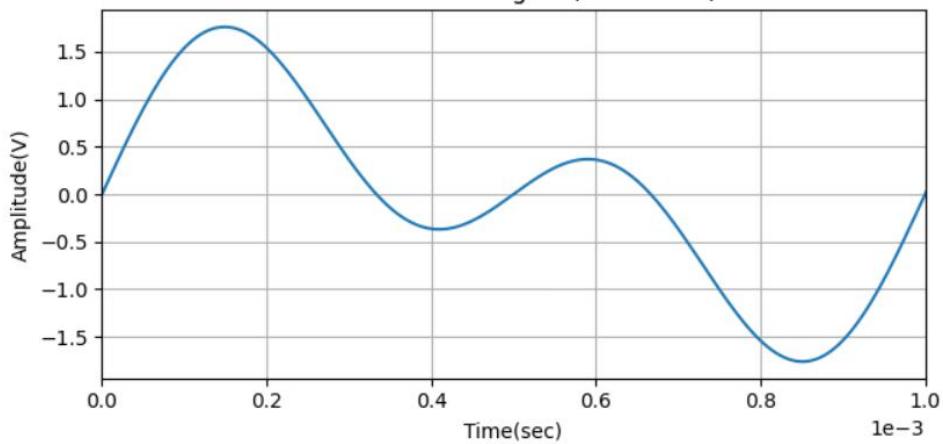
Sampled Signal ($f_s = 3.0\text{kHz}$)Reconstructed Signal ($f_s = 3.0\text{kHz}$)Sampled Signal ($f_s = 1.9\text{kHz}$)Reconstructed Signal ($f_s = 1.9\text{kHz}$)

Όπως φαίνεται από το παραπάνω διάγραμμα, παρόλο που τα δείγματα με $f_s=4f_m$ είναι πολύ λιγότερα από αυτά των δειγματοληψιών με $f_s=30f_m$ / $f_s=30f_m$ μπορούμε να ανακατασκευάσουμε το δοθέν σήμα. Σύμφωνα με το Θεώρημα Δειγματοληψίας η ελάχιστη θεωρητική f_s ώστε να είναι δυνατή η ακριβής ανακατασκευή του σήματος υπό συζήτηση είναι $f_{s(\min)} = 2f_B = 2\text{kHz}$. ($F\{\sin(2000\pi t)\}=\pi \cdot j[\delta(\omega+\omega_0)-\delta(\omega-\omega_0)]$) Προσεγγίζοντας το θέμα πρακτικά, δοκιμάζουμε διαφορετικές συχνότητες δειγματοληψίας και προσπαθούμε να ανακατασκευάσουμε το αρχικό σήμα. Παρατηρούμε ότι για συχνότητες $f_s \geq 2f_m$ (π.χ. για $f_s=3\text{kHz}$) το σήμα ανακατασκευάζεται επιτυχώς. Ωστόσο παρατηρούνται μικρές παραμορφώσεις που αφορούν το πλάτος του, λόγω της προσέγγισης του άπειρου αθροίσματος (του θεωρήματος δειγματοληψίας) με το αντίστοιχο πεπερασμένο. Για $f_s < 2f_m$ παρατηρούμε ότι το ανακατασκευασμένο σήμα έχει μικρότερη συχνότητα από αυτή του αρχικού σήματος, έχουμε δηλαδή εμφανής παραμόρφωση του φάσματος.

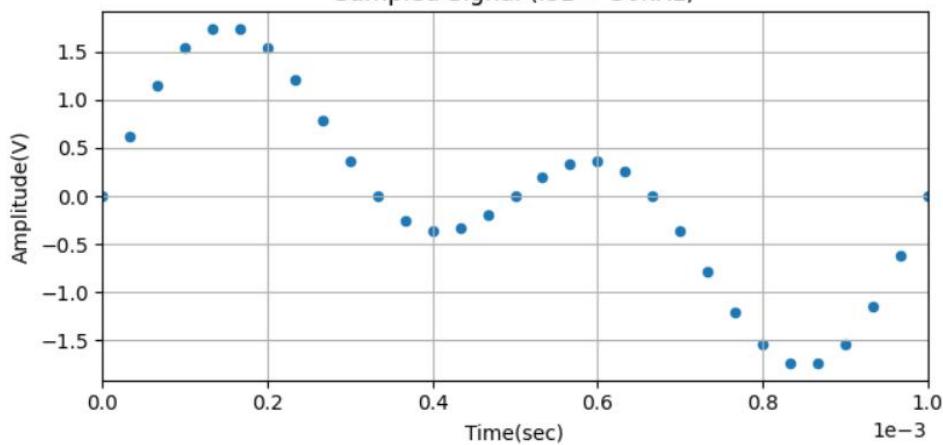
Συνεπώς, και υπό πρακτική προσέγγιση, η ελάχιστη συχνότητα ανακατασκευής σήματος είναι $f_{s(\min)}=2\text{kHz}$.

(ii'): Έχουμε $q(t) = \sin(2\pi \cdot 1000t) + \sin(2\pi \cdot 2000t)$. Το σήμα που προκύπτει είναι η υπέρθεση δύο ημιτονοειδών συναρτήσεων με συχνότητες $f_1 = 1\text{kHz}$ και $f_2 = 2\text{kHz}$. Για τη συχνότητα του $q(t)$ ισχύει $T' = \text{LCM}(T_1, T_2)$, δηλαδή $T' = T_m = 1\text{ms}$ και $f' = f_m = 1\text{kHz}$. Επαναλαμβάνουμε τα ερωτήματα α', β' για το σήμα $q(t)$. Για να αποφύγουμε περιττές επαναλήψεις του κώδικα, παραμετροποιούμε κατάλληλα τον κώδικα των προηγούμενων ερωτημάτων έτσι ώστε να εφαρμόζει κατάλληλα σε όλες τις ζητούμενες περιπτώσεις. Στα διαγράμματα αυτού του υποερωτήματος παρουσιάζεται μία περίοδος. Άρα έχουμε:

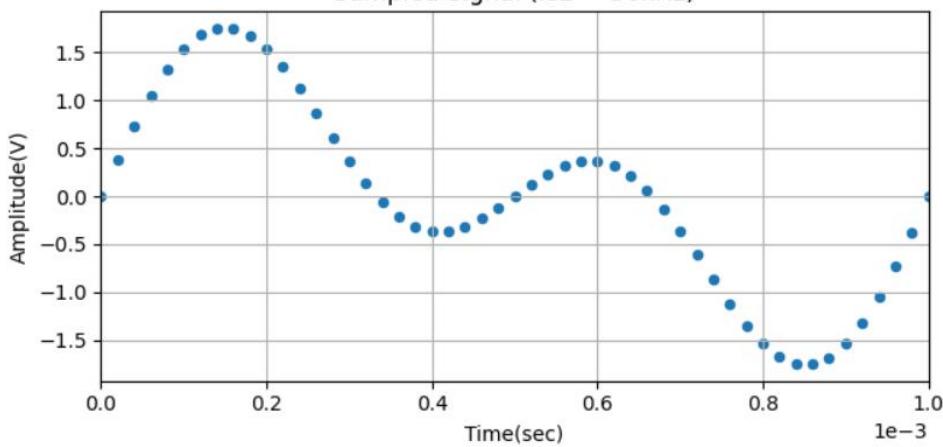
Given Periodic Signal (fm = 1kHz)



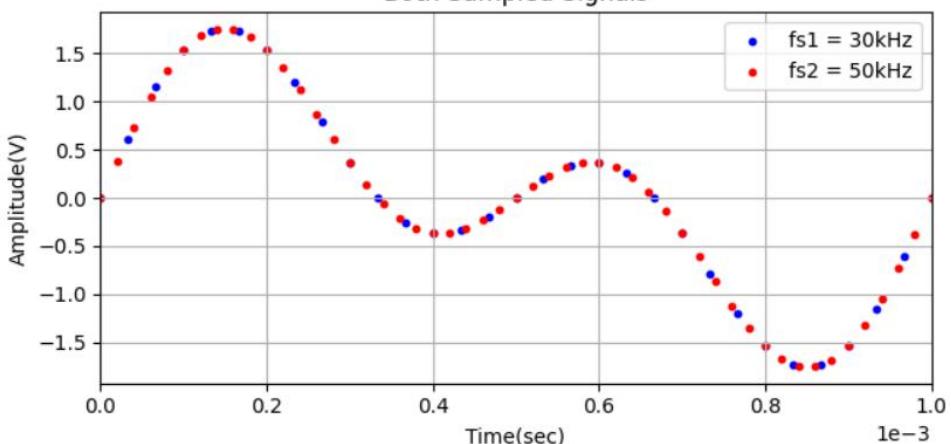
Sampled Signal (fs1 = 30kHz)

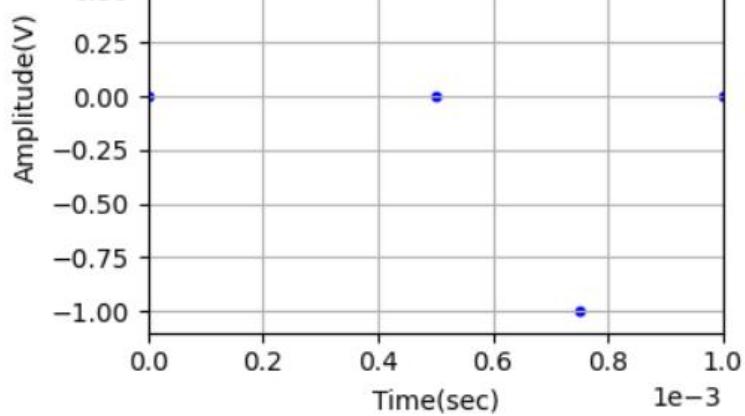
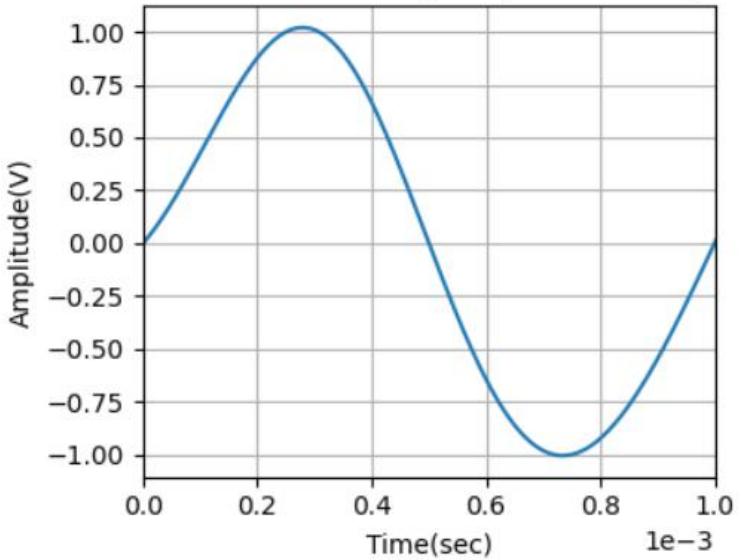
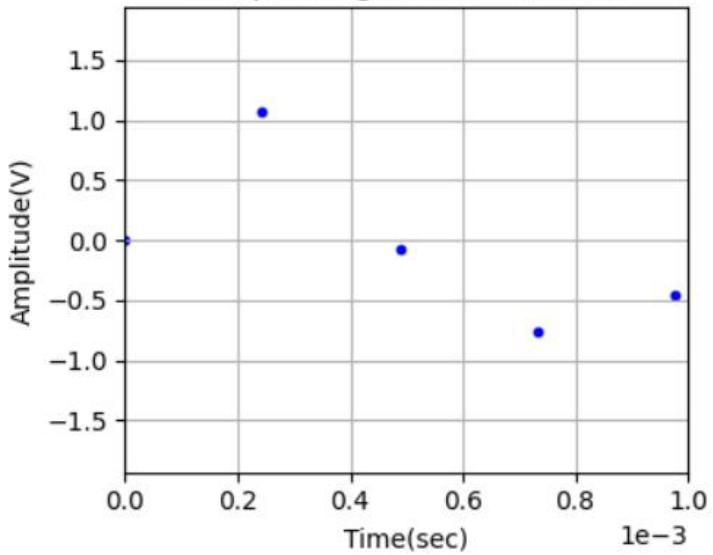
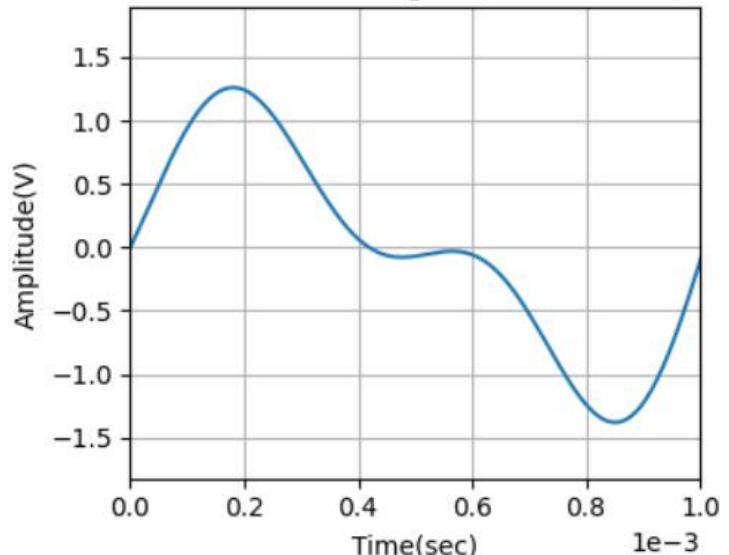
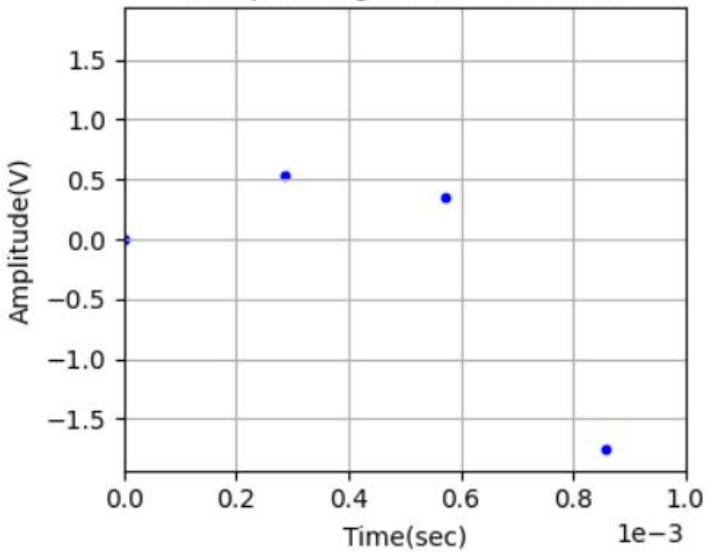
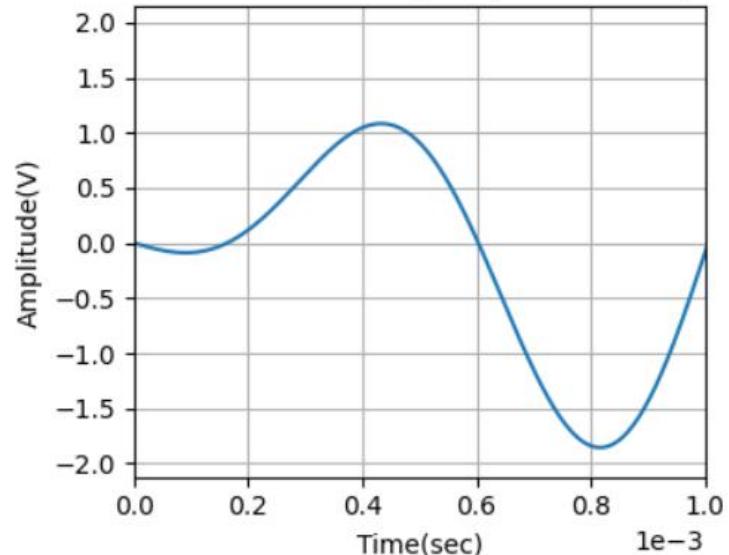


Sampled Signal (fs2 = 50kHz)



Both Sampled Signals



Sampled Signal ($fs = 4.0\text{kHz}$)Reconstructed Signal ($fs = 4.0\text{kHz}$)Sampled Signal ($fs = 4.1\text{kHz}$)Reconstructed Signal ($fs = 4.1\text{kHz}$)Sampled Signal ($fs = 3.5\text{kHz}$)Reconstructed Signal ($fs = 3.5\text{kHz}$)

Όπως φαίνεται από το παραπάνω διάγραμμα, για $f_s=4f_m$ δεν μπορούμε να ανακατασκευάσουμε το δοθέν σήμα, καθώς απομονώνουμε μόνο τη μία συχνοτική συνιστώσα. Σύμφωνα με το Θεώρημα Δειγματοληψίας $f_s > 2f_B = 4\text{kHz}$, συνεπώς (αφού δεν ισχύει η ισότητα) δεν υπάρχει ελάχιστη θεωρητική f_s ώστε να είναι δυνατή η ακριβής ανακατασκευή του σήματος υπό συζήτηση. Μπορεί να γίνει επιλογή συχνότητας δειγματοληψίας ελαφρώς υψηλότερη των 4kHz ώστε να υπάρχει η δυνατότητα ανακατασκευής, όπως φαίνεται παραπάνω. Προσεγγίζοντας το θέμα πρακτικά, δοκιμάζουμε διαφορετικές συχνότητες δειγματοληψίας και προσπαθούμε να ανακατασκευάσουμε το αρχικό σήμα.

Το φάσμα του δοθέντος σήματος περιέχει δύο αρμονικές συχνοτικές συνιστώσες $f_1 = 1\text{kHz}$, $f_2 = 2\text{kHz}$. Έχουμε $f_B = 2\text{kHz}$ και παρατηρούμε ότι για συχνότητες $f_s > 2f_B$ (π.χ. $f_s=4.1\text{kHz}$) το σήμα ανακατασκευάζεται επιτυχώς. Ωστόσο παρατηρούνται μικρές παραμορφώσεις που αφορούν το πλάτος του, λόγω της προσέγγισης του άπειρου αθροίσματος (του θεωρήματος δειγματοληψίας) με το αντίστοιχο πεπερασμένο. Επίσης, και για $f_s \leq 2f_m$ παρατηρούμε ότι το ανακατασκευασμένο σήμα έχει μικρότερη συχνότητα από αυτή του αρχικού σήματος, έχουμε δηλαδή εμφανής παραμόρφωση του φάσματος. Συνεπώς, και υπό πρακτική προσέγγιση, η ελάχιστη συχνότητα ανακατασκευής σήματος προσεγγίζει την $2f_B$.

Συνεπώς παρατηρούμε ότι το Θεώρημα Δειγματοληψίας μερικές φορές συμπεριλαμβάνει την συχνότητα $2f_B$ στις αποδεκτές και μερικές όχι. Εξαρτάται από την ύπαρξη αρμονικής συνιστώσας στη συχνότητα f_B .

Ο τελικός κώδικας σε Python για το 1^ο Ερώτημα:

```

1 # Raptopoulos Petros el19145, Team 11 (No Partner)
2 # import necessary Libraries
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from numpy import pi as pi
6 import math
7
8 # Exercise 1
9 # global declared constants
10 fm = 1000 # f = 1kHz (1+4+5=10, 1+0=1)
11 T = 0.001 # T = 10^(-3)sec
12 conStep = 0.000005 # Step small enough to consider
13 # the outcome continuous
14 t = np.arange(0,4*T, conStep) # define time axis
15 # span the time values over 4 periods
16 # we consider the time axis continuous despite
17 # its discrete representation
18
19 def sine(t, f = fm): # define sine function for each time point
20     return np.sin(2*pi*f*t)
21
22 def sinc(x): # define sinc function for each time point
23     if x == 0:
24         return 1
25
26     else:
27         return math.sin(pi*x)/(pi*x)
28
29 def tri(t): # define triangular function for each time point
30     t = t % 0.001 # find the mod in order to apply
31 # formulas for the first pulse
32     if 0 <= t and t <= 0.0005: # 0 <= t <= T/2
33         return 16000*t**4
34     elif 0.0005 <= t and t <= 0.001: # T/2 <= t <= T
35         return -16000*t+12
36
37 vtri = np.vectorize(tri)
38 # make the tri() function an element-wise one
39
40 def q(t):
41     return sine(t) + sine(t, fm+1000)
42 # fm = 1kHz, A = 1kHz, A = 1V
43
44 funct = [ "", vtri, sine, q ]
45 # List the functions used in each iteration
46 # start the functions from index 1
47 for iterNum in range(1,4):
48     # a for loop to avoid code repetition
49     # question (c'),(b'): iterNum = 1
50
51     # question (c'.ii): iterNum = 2
52     # question (c'.iii): iterNum = 3
53
54     if iterNum != 3:
55         xAxisMax = 4*T
56         # start from 0 and end just before 4*T
57     else:
58         xAxisMax = T
59         # start from 0 and end just before T
60         # for (c.ii) we want one period to be plotted
61         # four periods otherwise
62
63     y = funct[iterNum](t) # the continuous signal
64     # y_i = funct(t_i), for all i
65
66     fig1 = plt.figure(2*iterNum-1) # create a new figure
67     # first iteration uses fig 1,2 , second call uses fig 3,4 ...
68     ax = fig1.add_subplot(221) # create one subplot
69     plt.xlabel("Time(sec)")
70     plt.ylabel("Amplitude(V)")
71     plt.title("Given Periodic Signal (fm = 1kHz)")
72     plt.grid() # use the grid
73     ax.plot(t,y)
74     ax.set_xlim(left = 0, right = xAxisMax) # start from 0 and end

```

```

75 # just before xAisMax
76
77 plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
78 # use the scientific (exponential) notation for the x axis
79
80 # Question (a')
81 # (i)
82
83 # first sampling process
84 fs1 = 30*fm
85 Ts1 = 1/fs1
86 t1 = np.arange(0,4*T+Ts1, Ts1) # define time axis
87 # the instruction np.arange() isn't end-inclusive
88 # we must add Ts1 to the end in order to include
89 # the sample value at t = 4*T
90
91 y1 = funct[iterNum](t1) # y_i = funct(t_i), for all i
92
93 ax1 = fig1.add_subplot(222) # create one subplot
94 plt.xlabel("Time(sec)")
95 plt.ylabel("Amplitude(V)")
96 plt.title("Sampled Signal (fs1 = 30kHz)")
97 plt.grid() # use the grid
98 ax1.scatter(t1,y1, s=20, marker='o') # adjust size and marker

```

```

97     ax1.set_xlim(left = 0, right = xAxiMax) # start from 0 and end
98             # just before xAxiMax
99
100    plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
101    # use the scientific (exponential) notation for the x axis
102
103    # (ii)
104
105    # second sampling process
106    fs2 = 50*fm
107    Ts2 = 1/fs2
108    t2 = np.arange(0,4*T+Ts2, Ts2) # define time axis
109    # the instruction np.arange() isn't end-inclusive
110    # we must add Ts1 to the end in order to include
111    # the sample value at t = 4*T
112
113    y2 = funct[iterNum](t2) # y_i = funct(t_i), for all i
114
115    ax2 = fig1.add_subplot(223) # create one subplot
116    plt.xlabel("Time(sec)")
117    plt.ylabel("Amplitude(V)")
118    plt.title("Sampled Signal (fs2 = 50kHz)")
119    plt.grid() # use the grid
120    ax2.scatter(t2,y2, s=20, marker='o') # adjust size and marker
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
```

Question (b')

```

145 if iterNum != 3:
146     fs = [4*fm, 3*fm, 1.9*fm]
147     # 3 sample frequencies for all questions except (c.ii)
148 else:
149     fs = [4*fm, 4.1*fm, 3.5*fm]
150 fig2 = plt.figure(2*iterNum) # create a new figure
151 for j in range(0,3): # j = 1 for fs = 4*fm ...
152 # Using a for Loop to avoid code repetitions
153     Ts = 1/fs[j]
154     ts = np.arange(0,400*T+Ts, Ts) # define time axis
155     # for each sample frequency
156
157     ys = funct[iterNum](ts) # ys_i = funct(ts_i), for all i
158     ax1 = fig2.add_subplot(2, 3, 1+j) # create one subplot
159     plt.xlabel("Time(sec)")
160     plt.ylabel("Amplitude(V)")
161     plt.title("Sampled Signal (fs = {}kHz)".format(fs[j]/1000))
162     plt.grid() # use the grid
163     ax1.scatter(ts,ys, s=10, c='b', marker='o')
164     # adjust size, marker, color and label
165     ax1.set_xlim(left = 0, right = xAxiMax)
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
```

```

193     fig2.tight_layout() # adjust the spacing between subplots in
194             # order to avoid text overlapping
195
196 plt.show() # show the figures and their plots

```

```

121     ax2.set_xlim(left = 0, right = xAxiMax) # start from 0 and end
122             # just before xAxiMax
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
```

```

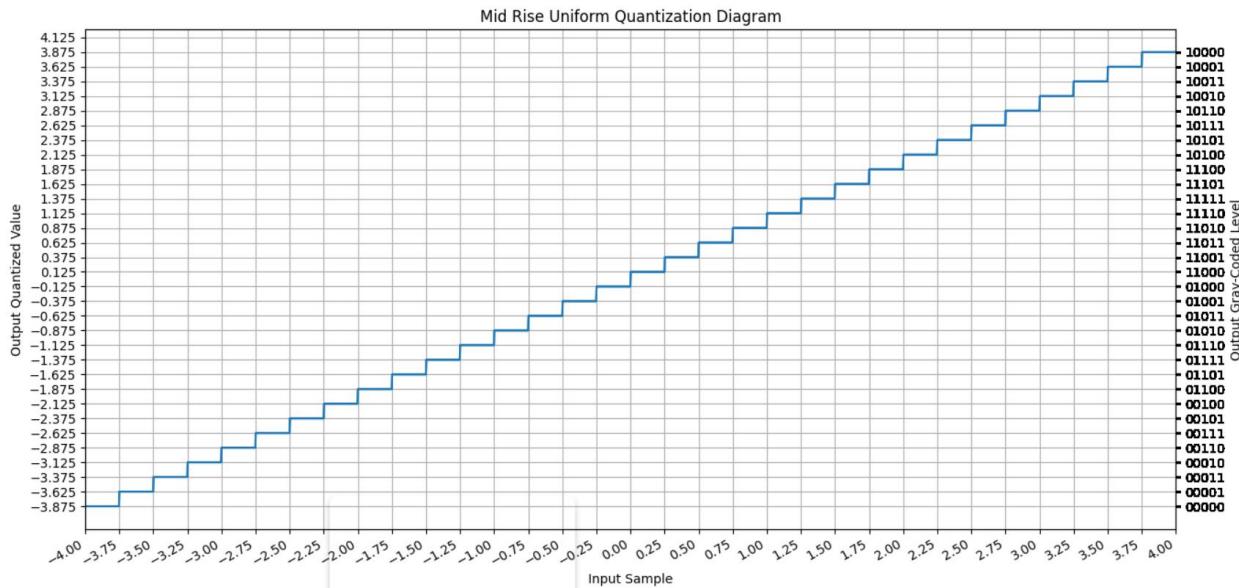
# start from 0 and end just before xAxiMax
plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
# use the scientific (exponential) notation for the x axis

# Try to reconstruct the signal from its samples
yrecon = np.empty(t.size)
for i in range(t.size):
    yrecon[i] = 0
    for n in range(ys.size):
        yrecon[i] += ys[n]*sinc((i*conStep-n*Ts)/Ts)
# From the theory we have y(t)=sum(y[nTs])sinc((t-nTs)/Ts)
# for each n (integer) from -00 to +00
ax2 = fig2.add_subplot(2, 3, 4+j) # create one subplot
plt.xlabel("Time(sec)")
plt.ylabel("Amplitude(V)")
plt.title("Reconstructed Signal (fs = {})kHz".format(fs[j]/1000))
plt.grid() # use the grid
ax2.plot(t,yrecon) # plot the reconstructed signal
ax2.set_xlim(left = 0, right = xAxiMax) # start from 0 and end
# just before xAxiMax
plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
# use the scientific (exponential) notation for the x axis

```

2^ο Ερώτημα:

(α'): Θεωρούμε είσοδο σε ομοιόμορφο κβαντιστή 5 bits (mid riser) το σήμα $y(t)$ του 1^{ου} ερωτήματος μετά από δειγματοληψία συχνότητας $f_{si} = 30f_m$. Γνωρίζουμε ότι ο κβαντισμός (quantization) πλάτους ορίζεται ως η διαδικασία μετατροπής του πλάτους του δείγματος $y(nTs)$ του σήματος πληροφορίας $y(t)$ στη χρονική στιγμή $t = nTs$ σε ένα διακριτό πλάτος $v(nTs)$ το οποίο λαμβάνεται από ένα πεπερασμένο σύνολο δυνατών τιμών πλάτους. Συγκεκριμένα για ένα ομοιόμορφο κβαντιστή (mid riser), σύμφωνα με τη θεωρία, η σχέση κβαντισμού είναι $Q(y) = \Delta \cdot (\left\lfloor \frac{y}{\Delta} \right\rfloor + \frac{1}{2})$, όπου $\Delta = \frac{2 \cdot y_{max}}{\text{levels}} = \frac{2 \cdot y_{max}}{2^{\text{bits}}}$. Έχουμε λοιπόν το διάγραμμα κβάντισης:



Στα δεξιά του διαγράμματος φαίνεται η ανάθεση κώδικα αναπαράστασης Gray σε κάθε επίπεδο κβάντισης, με το "χαμηλότερο" επίπεδο να θεωρείται το μηδενικό. Γνωρίζουμε ότι ο κώδικας Gray είναι μια διάταξη του δυαδικού συστήματος αριθμών έτσι ώστε δύο διαδοχικές τιμές να διαφέρουν μόνο σε ένα δυαδικό ψηφίο. Συνεπώς το παραπάνω διάγραμμα αναπαριστά την έξοδο του κβαντιστή (με κωδικοποίηση Gray) για κάθε τιμή εισόδου.

Υλοποίηση Κωδικοποίησης Gray σε Python

Υλοποίηση Κβάντισης σε Python

```

65 def quantize(x): # apply classification rule
66     temp = np.floor(x/D)
67     return temp-1 if temp == levels/2 else temp
68     # Level nums for negative sample-values = -16...-1
69     # Level nums for positive sample-values = 0...15
70     # we notice that the 15th Level is overloaded
71     # as the classification rule np.floor(x/D) for
72     # x = +m_max gives 16, which is rejected.
73     # (we have 2^bits Levels not 2^bits+1)
74     # so we assign for x = +m_max the 15th level
75
76 vquantize = np.vectorize(quantize)
77 # make the quantize() function an element-wise one
78
79 # plot the quantization diagram
80 In = np.linspace(-4, 4, 3000)
81 qOut = D*(vquantize(In) + 1/2)
82 # we have a mid rise uniform quantizer
83 # according to the theory the above formula
84 # must be used
85
86 fig = plt.figure(7) # create a new figure
87 ax1 = fig.add_subplot(111) # create one subplot
88 ax2 = ax1.twinx() # create a second y axis

```

```

46 def gray(x): # x: input int, output binary gray code
47     x = int(x) # must do this because some values
48                     # are stored as floats (2.) and the
49                     # bitwise operations dont work
50     return "{0:b}".format(x^(x>>1)).zfill(5)
51 # return the gray code as a string consisting of 5
52 # binary digits. ^ -> bitwise xor, >> -> shift right
53
54 vgray = np.vectorize(gray)
55 # make the gray() function an element-wise one
56
57 num = -4
58 xticks = []
59 while(num <= 4):
60     xticks.append(num)
61     num += D
62
63 plt.xticks(xticks) # plot ticks of x axis
64 plt.title("Mid Rise Uniform Quantization Diagram")
65 ax2.plot(In, qOut)
66 plt.ylabel("Output Gray-Coded Level")
67 ax1.set_xlabel("Input Sample")
68 ax1.set_ylabel("Output Quantized Value")
69 ax1.set_yticks(np.array(xticks) + D/2) # ticks of first y axis
70 ax1.plot(In, qOut)
71 ax1.grid() # use the grid for the first axis
72 ax1.set_xlim(left = -4, right = 4) # set axis limits
73 y2ticks = vgray(vquantize(In) + levels/2)
74 plt.yticks(qOut, y2ticks) # plot ticks for second y axis
75
76 fig.autofmt_xdate() # rotate xticks to avoid text overlapping

```

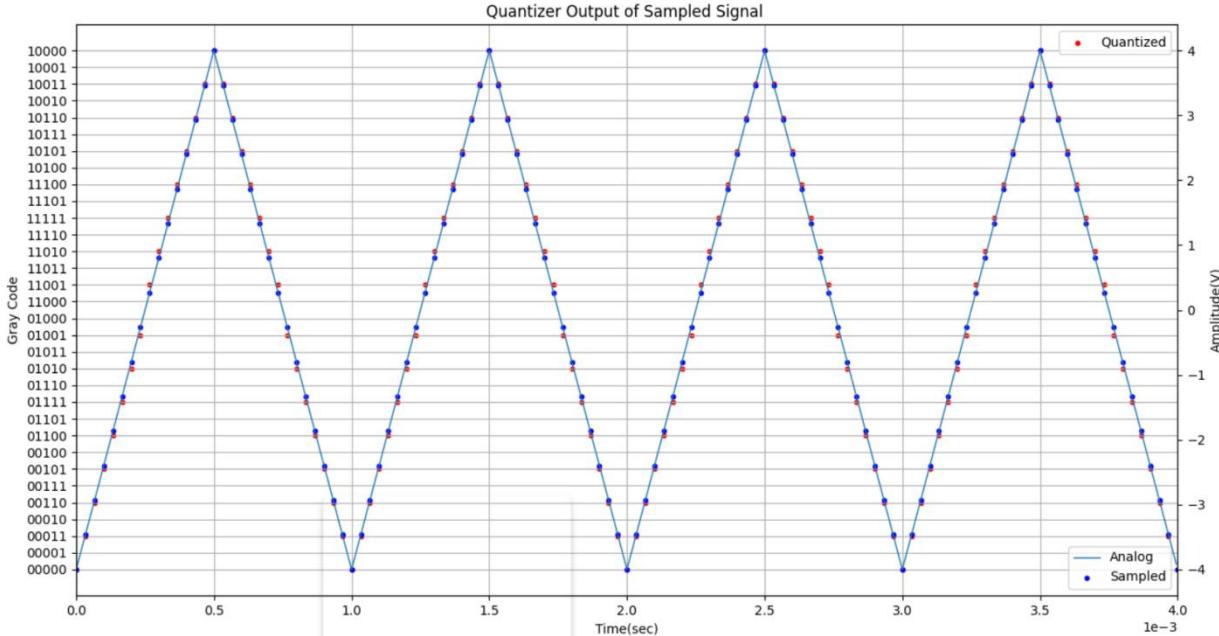
Υλοποίηση Διαγράμματος

Θεωρώντας είσοδο το δειγματοληπτούμενο σήμα στον ομοιόμορφο (mid-rise) κβαντιστή έχουμε την παρακάτω έξοδο (κόκκινο χρώμα) από τον κβαντιστή. (Για σύγκριση παρουσιάζονται τα δείγματα της δειγματοληψίας για $f_{si} = 30f_m$ αλλά και το σήμα $y(t)$)

```

57  fs1 = 30*fm
58  Ts1 = 1/fs1
59  ts = np.arange(0,4*T+Ts1, Ts1)
60  ys = vtri(ts) # sampling process
61  bits = 5 # fm = 1kHz odd
62  levels = 2**bits # Levels of quantization
63  D = 2*4/levels # quantization step = 2*maxValue/levels

```



(β'): Παρατηρούμε ότι οι κβαντισμένες τιμές παρουσιάζουν απόκλιση από τις αντίστοιχες δειγματοληπτούμενες. Ονομάζουμε λοιπόν σφάλμα κβάντισης την διαφορά της i -οστής κβαντισμένης τιμής με την αντίστοιχη i -οστή τιμή δειγματοληψίας. ($\text{error}_i = q_i - s_i$)

Για τον αριθμητικό μέσο μ των σφάλματος κβάντισης έχουμε: $\mu = \frac{1}{N} \sum_{n=0}^{N-1} \text{error}_i$

Για την τυπική απόκλιση σ του σφάλματος κβάντισης έχουμε: $\sigma^2 = \frac{1}{N} \sum_{n=0}^{N-1} (\text{error}_i - \mu)^2$

Υλοποιούμε τα παραπάνω σε Python

(όπως φαίνεται στην εικόνα στα δεξιά) και υπολογίζουμε την τυπική απόκλιση του σφάλματος κβάντισης:

(i): για τα 10 πρώτα δείγματα: ≈ 0.076376

(ii): για τα 20 πρώτα δείγματα: ≈ 0.072648

(iii): Υπολογίζουμε το SNR κβάντισης, σύμφωνα με τις σημειώσεις του μαθήματος:

$$SNR_q = \frac{\text{Tυπική Απόκλιση Δειγμάτων}}{\text{Tυπική Απόκλιση Σφάλματος}}$$

Άρα **(i)**: $SNR_q \approx 26.045$ (dB), **(ii)**: $SNR_q \approx 30.589$ (dB)

```

111  yqLvl = vquantize(ys) + levels/2 # y_quantization_Level
112  # we want to count the Levels from the bottom
113  fig = plt.figure(8) # create a new figure
114  ax3 = fig.add_subplot(111) # create one subplot
115  plt.yticks([i for i in range(0,levels)], [gray(i) for i in range(32)])
116  # plot ticks for y axis (gray code)
117  plt.title("Quantizer Output of Sampled Signal ")
118  ax3.scatter(ts,yqLvl, s=10, c='r', marker='o', label='Quantized')
119  ax3.legend(loc = 'upper right') # show the Legends on the plot
120  ax4 = ax3.twinx() # create a second y axis
121  # but on the same plot
122  ax4.scatter(ts,ys, s=10, c='b', marker='o', label='Sampled')
123  ax4.plot(t,vtri(t), linewidth=1, label='Analog')
124  ax4.legend(loc = 'lower right') # show the Legends on the plot
125  ax3.set_ylabel("Amplitude(V)")
126  ax3.set_xlabel("Time(sec)")
127  ax3.set_ylabel("Gray Code")
128  ax3.grid() # use the grid
129  ax3.set_xlim(left = 0, right = 4*T) # start from 0 and end
130  # just before 4*T
131  ax4.set_xlim(left = 0, right = 4*T) # start from 0 and end
132  # just before 4*T
133  plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
134
135  yq = D*(vquantize(ys) + 1/2) # output values of quantizer
136  error = yq - ys # quantization error
137  def mean(itemsNum, array): # return arithmetic mean
138      sum = 0
139      for i in range(itemsNum):
140          sum += array[i]
141      return sum/itemsNum
142  def stdDev(itemsNum, array): # return Standard Deviation
143      return (mean(itemsNum, (array - mean(itemsNum, array))**2))**(1/2.0)
144  #  $\sigma^2 = E[(\text{error}_i - \mu)^2]$ 
145
146  print("Standard Deviation (10 samples):", stdDev(10, error))
147  print("Standard Deviation (20 samples):", stdDev(20, error))
148  def SNRq(itemsNum): # return Signal-Noise Ration
149      return (stdDev(itemsNum, ys)/stdDev(itemsNum, error))**2
150  print("SNRq (10 samples):", SNRq(10),"in dBs:", 10*math.log10(SNRq(10)))
151  print("SNRq (20 samples):", SNRq(20),"in dBs:", 10*math.log10(SNRq(20)))
152  print("SNRq (30 samples):", SNRq(30),"in dBs:", 10*math.log10(SNRq(30)))

```

Υλοποίηση Διαγράμματος

Υλοποίηση Τυπικής Απόκλισης

Υπολογίζουμε τη θεωρητική τιμή SNR κβάντισης σύμφωνα με τις σημειώσεις του μαθήματος:

$$SNR_q = \left(\frac{3P}{y_{max}^2} \right) \cdot 2^{2 \cdot bits}$$

Standard Deviation (first 10 samples): 0.07637626158259737
Standard Deviation (first 20 samples): 0.0726483157256779
SNRq (first 10 samples): 402.285714285714 ,in dBs: 26.045346104558178
SNRq (first 20 samples): 1145.2631578947367 ,in dBs: 30.58905290073313
SNRq (first 30 samples): 1024.0 ,in dBs: 30.102999566398122

Αποτελέσματα τυπωμένα στο Terminal

Όπου P η μέση ισχύς του σήματος πληροφορίας.

$$\text{Έχουμε: } P = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} y(t)^2 dt = \frac{1}{T} \left(\int_{-\frac{T}{2}}^0 (-16000t - 4)^2 dt + \int_0^{\frac{T}{2}} (16000t - 4)^2 dt \right) = \dots = \frac{16}{3}$$

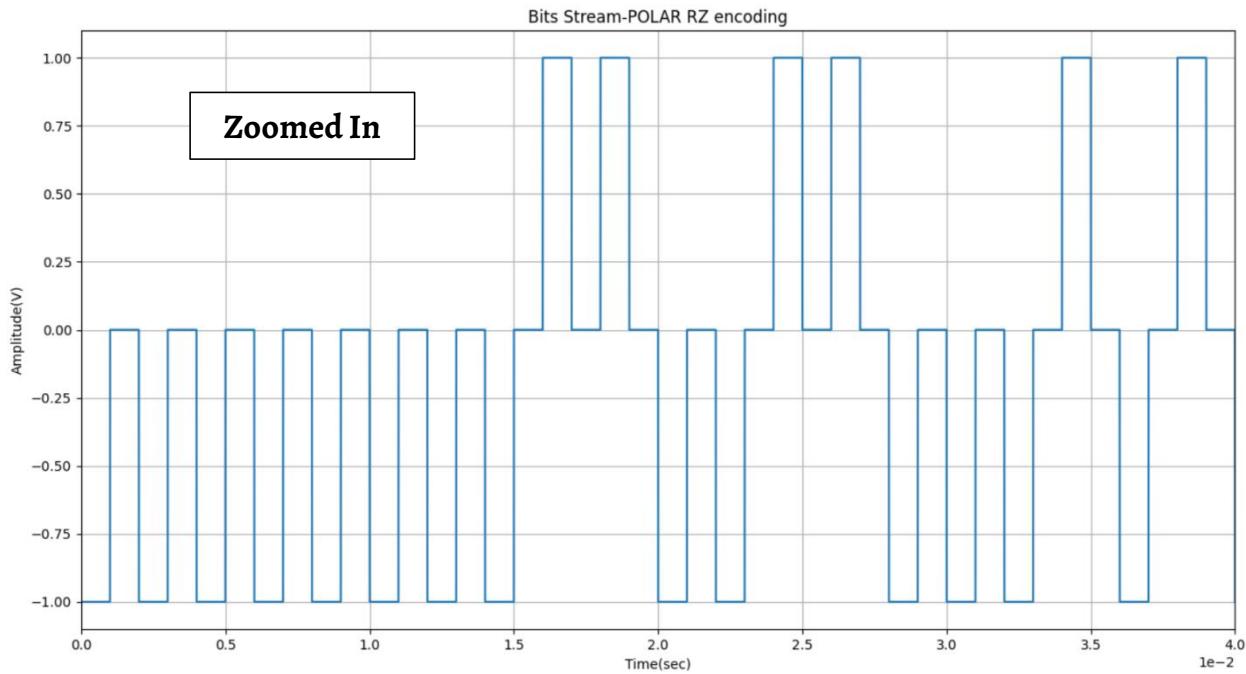
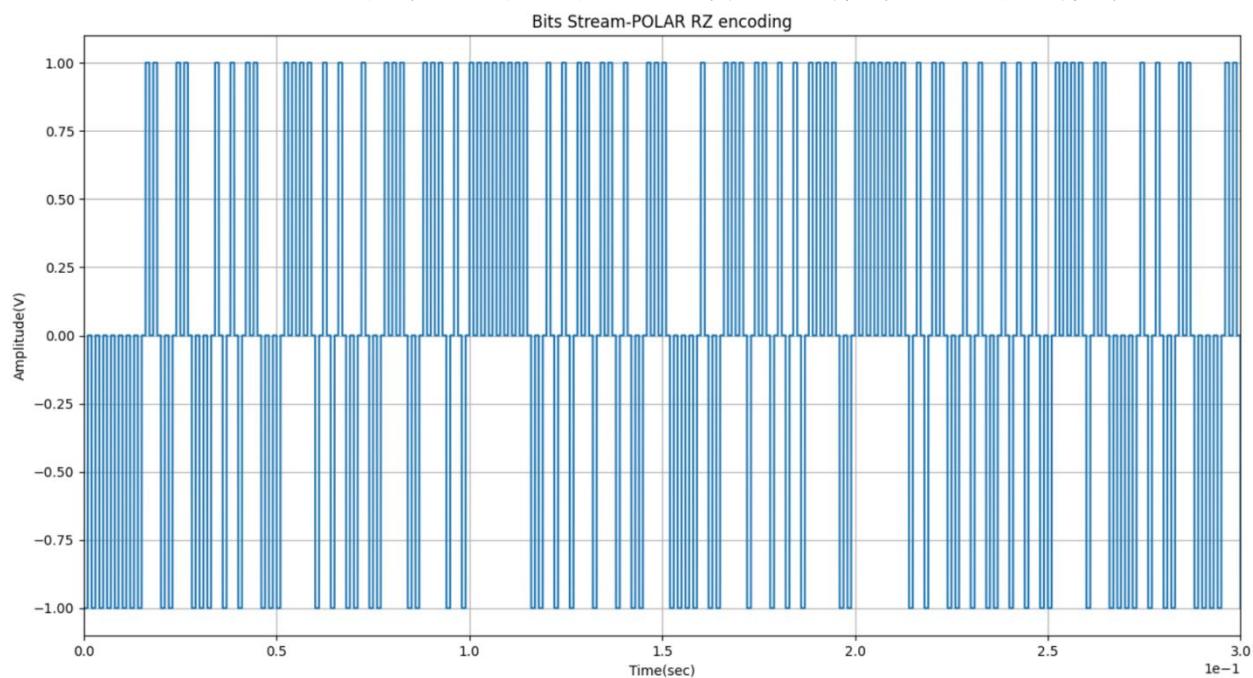
Άρα θεωρητικό $SNR_q = 2^{10}$, δηλαδή $SNR_q(dB) = 10 \log(SNR_q) = 10 \log(2^{10}) = 100 \log 2 \approx 30.1$

Για να είναι ο υπολογισμός του σηματοθορυβικού λόγου αξιόπιστος πρέπει να γίνει βασισμένος στα πρώτα 30 δειγμάτα (αφού έχουμε $f_{SI} = 30f_m$). Έτσι, όπως φαίνεται παραπάνω, το SNR κβάντισης των 30 δειγμάτων ισούται με τη θεωρητική αναμενόμενη τιμή, αφού καλύπτει πλήρως μία περίοδο αναλογικού σήματος.

Αντίθετα το SNR κβάντισης των 10/20 δειγμάτων διαφέρει αρκετά από τη θεωρητική τιμή. Αυτό συμβαίνει διότι δεν έχουν συμπεριλάβει επαρκώς τα χαρακτηριστικά ολόκληρης της περιόδου του σήματος.

(γ'): Παρουσιάζουμε σε διάγραμμα για μια περίοδο την ροή μετάδοσης από bits μετά την κβάντιση, θεωρώντας κωδικοποίηση γραμμής POLAR RZ με διάρκεια bit 2msec και A = 1V.

Στη κωδικοποίηση γραμμής POLAR RZ για τη πρώτη μισή διάρκεια συμβόλου έχουμε +AV αν το εν λόγω bit είναι 1 ενώ -AV αν είναι 0. Στο δεύτερο μισό της διάρκειας συμβόλου έχουμε 0V. Άρα έχουμε:



```

158 yc = vgray(yqlvl) # y_coded
159 Tc = 0.002 # duration of a bit
160 tc = np.linspace(0,bits*Tc*(fs1/fm), 10000)
161 # we want to plot only one period
162 def polarRZ(t):
163     bitsCount = int(t/Tc)           # count all bits encoded
164                         # starting from zero
165     bitIndex = bitsCount % bits   # at which bit are we inside
166                         # one gray code
167     codeIndex = int(bitsCount/bits) # at which gray code are we
168                         # inside yc
169     if 1/2 <= t/Tc - bitsCount and t/Tc - bitsCount < 1:
170         return 0
171     else:
172         return 1 if yc[codeIndex][bitIndex] == '1' else -1
173
174 vpolarRZ = np.vectorize(polarRZ)
175 # make the quantize() function an element-wise one
176
177 fig = plt.figure(9) # create a new figure
178 ax = fig.add_subplot(111) # create one subplot
179 plt.title("Bits Stream-POLAR RZ encoding")
180 ax.plot(tc, vpolarRZ(tc))
181 plt.ylabel("Amplitude(V)")
182 plt.xlabel("Time(sec)")
183 plt.grid() # use the grid
184 ax.set_xlim(left = 0, right = bits*Tc*(fs1/fm))
185 plt.ticklabel_format(axis="x", style="sci", scilimits=(-3,3))

```

Υλοποίηση σε Python

3^o Ερώτημα:

(α'): Παράγουμε τυχαία ακολουθία 36 ψηφίων (bits) με ίση πιθανότητα εμφάνισης τιμών ο ή 1:

[1 1 1 1 0 1 1 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 1 0 1 0]

(i): Γράφουμε την προκύπτουσα ακολουθία συμβόλων για κάθε διαμόρφωση κωδικοποιώντας κατά Gray:

- ✓ BPSK: [1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0]
- ✓ QPSK: [11, 11, 01, 11, 10, 01, 00, 00, 01, 01, 00, 11, 00, 11, 11, 10, 10]
- ✓ 8PSK: [111, 101, 111, 001, 000, 000, 010, 100, 110, 011, 111, 010]

(ii): Για $T_b = 0.25s$, $f_c = 2Hz$, $A = 1V$ παρουσιάζουμε την κυματομορφή μετάδοσης για κάθε διαμόρφωση:

BPSK:

Symbol Gray Mapping	
0	s_0
1	s_1

(β'): Παρατίθεται για σύγκριση η διαμόρφωση BPAM της δυαδικής ακολουθίας. ($A = 1V$)

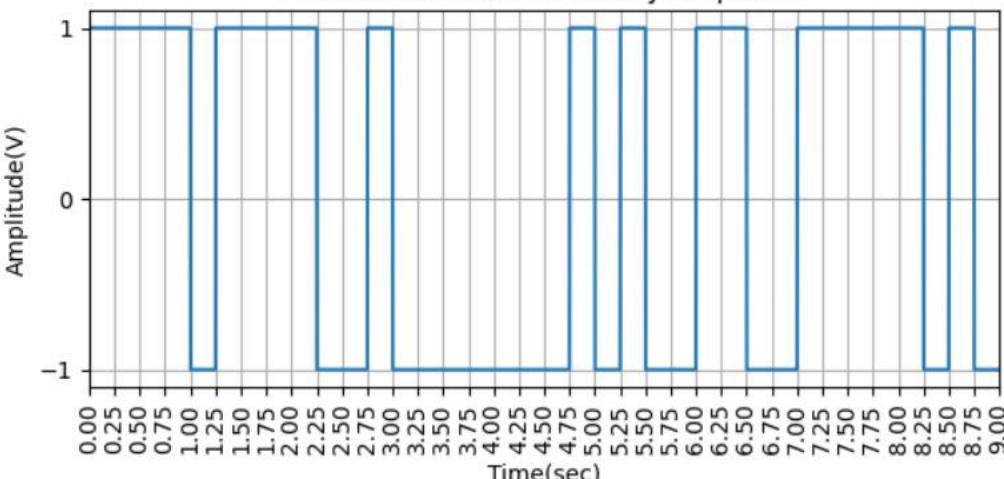
Κατά τη διαμόρφωση BPSK (Binary Phase Shift Keying) κάθε σύμβολο αναπαρίσταται από 1 bit. Έχουμε τις εξής κυματομορφές για την αναπαράσταση των δυαδικών ψηφίων:

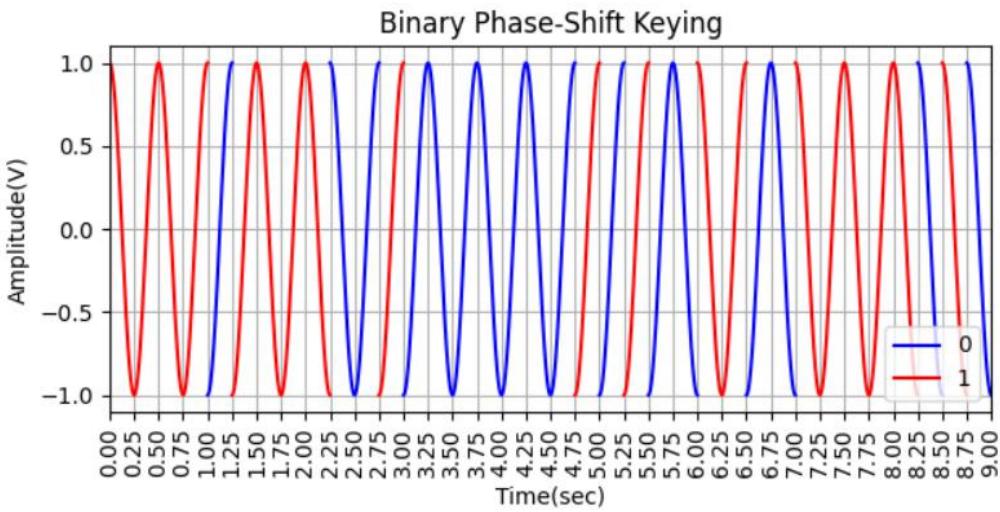
$$s_0(t): -\text{Acos}(2\pi f_c t), \text{ για bit } 0$$

$$s_1(t): \text{Acos}(2\pi f_c t), \text{ για bit } 1$$

(για $0 < t < T_b$)

B-PAM Modulation of Binary Sequence





```

188 bits = 36
189 bitSeq = np.random.randint(2, size = bits)
190 # generate random 36-bits sequence
191 # discrete uniform distribution
192 print(bitSeq) # print the sequence
193 Tb = 0.25 # symbol duration (sec)
194 fc = 2 # carrier frequency (Hz)
195 A = 1 # Amplitude (V)
196
197 t = np.linspace(0, bits*Tb, 10000, endpoint = False)
198 # consider continuous t, don't include t = bits*Tb
199 tb = np.arange(0, bits*Tb+Tb, Tb) # used for ticks
200
201 # Question (b')
202 def BPAM(t): # return amplitude, time used as input
203     return A if bitSeq[int(t/Tb)] == 1 else -A
204
205 vBPAM = np.vectorize(BPAM)
206 # make the vBPAM() function an element-wise one
207
208 fig = plt.figure(10) # create a new figure
209 ax1 = fig.add_subplot(221) # create one subplot
210 plt.title("B-PAM Modulation of Binary Sequence")
211 ax1.plot(t, vBPAM(t)) # plot BPAM signal
212 plt.ylabel("Amplitude(V)")
213 plt.xlabel("Time(sec)")
214 plt.grid() # use the grid
215 ax1.set_xlim(left = 0, right = bits*Tb)
216 ax1.set_xticks(tb) # set x axis' ticks
217 ax1.set_yticks([-1, 0, 1]) # set y axis' ticks

```

```

220 def BPSK_bit(t): # return current bit
221     return bitSeq[int(t/Tb)]
222 def BPSK(t): # which formula to use (gray coded)
223     if BPSK_bit(t) == 1:
224         return A*math.cos(2*pi*fc*t)
225     elif BPSK_bit(t) == 0:
226         return -A*math.cos(2*pi*fc*t)
227 vBPSK = np.vectorize(BPSK)
228 # make the BPSK() function an element-wise one
229 ax2 = fig.add_subplot(222) # create one subplot
230 plt.title("Binary Phase-Shift Keying")
231 prevBit = BPSK_bit(t[0])
232 lastPlotted = 0
233 # plot the color-Legends
234 ax2.plot([],[], c='b', label='0')
235 ax2.plot([],[], c='r', label='1')
236 colors = ['b','r']
237 for i in range(1, t.size):
238     if BPSK_bit(t[i]) != prevBit or i == t.size - 1:
239         ax2.plot(t[lastPlotted:i], vBPSK(t[lastPlotted:i]), c=colors[prevBit])
240         prevBit = BPSK_bit(t[i])
241         lastPlotted = i
242 # we plot zeros and ones with different colors
243 plt.ylabel("Amplitude(V)")
244 plt.xlabel("Time(sec)")
245 plt.grid() # use the grid
246 ax2.set_xlim(left = 0, right = bits*Tb)
247 ax2.set_xticks(tb)
248 ax2.legend(loc = 'lower right')
249 plt.setp(ax2.get_xticklabels(), rotation=90)
250 plt.setp(ax1.get_xticklabels(), rotation=90)

```

Τυλοποίηση σε Python

QPSK:

Symbol Gray Mapping

00	s_o
01	s_1
11	s_2
10	s_3

Κατά τη διαμόρφωση BPSK (Quadrature Phase Shift Keying) κάθε σύμβολο αναπαρίσταται από 2 bit. Έχουμε τις εξής κυματομορφές για την αναπαράσταση των δυαδικών ψηφίων:

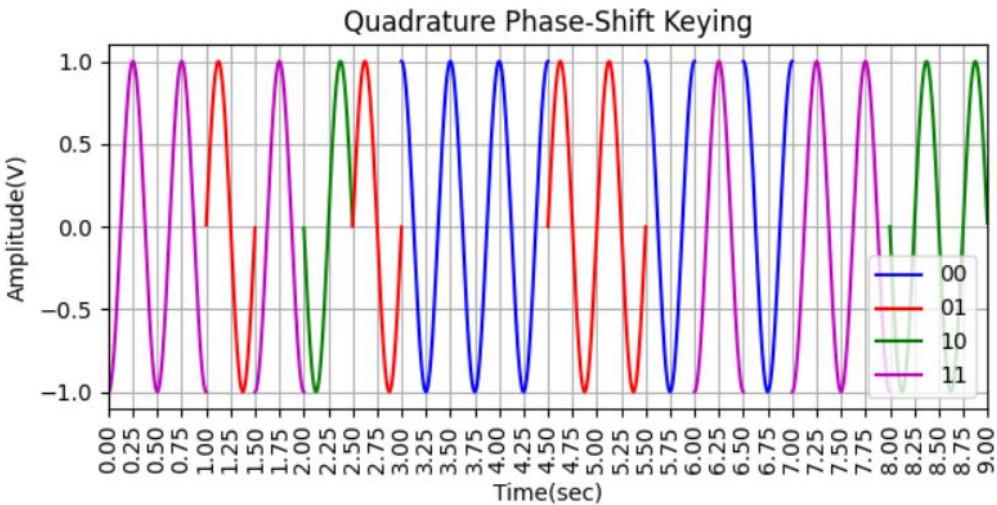
$$s_o(t): \text{Acos}(2\pi f_c t), \text{ για bits } 00$$

$$s_1(t): \text{Asin}(2\pi f_c t), \text{ για bit } 01$$

$$s_2(t): -\text{Acos}(2\pi f_c t), \text{ για bits } 11$$

$$s_3(t): -\text{Asin}(2\pi f_c t), \text{ για bit } 10$$

$$(\gamma \alpha \circ < t < 2T_b)$$



```

257 def QPSK_bits(t): # return current bits
258     return str(bitSeq[2*int(t/(2*Tb))])+str(bitSeq[2*int(t/(2*Tb))+1])
259
260 def QPSK(t): # which formula to use (gray coded)
261     if QPSK_bits(t) == '00':
262         return A*math.cos(2*pi*fc*t)
263     elif QPSK_bits(t) == '01':
264         return A*math.sin(2*pi*fc*t)
265     elif QPSK_bits(t) == '11':
266         return -A*math.cos(2*pi*fc*t)
267     elif QPSK_bits(t) == '10':
268         return -A*math.sin(2*pi*fc*t)
269
270 vQPSK = np.vectorize(QPSK)
271 # make the QPSK() function an element-wise one
272
273 ax3 = fig.add_subplot(223) # create one subplot
274 plt.title("Quadrature Phase-Shift Keying")
275 prevBits = QPSK_bits(t[0])
276 lastPlotted = 0

```

Υλοποίηση σε Python

```

277 colors = {'00':'b', '01':'r','10':'g','11':'m'}
278 for i in colors:
279     ax3.plot([],[], c=colors[i], label=i)
280 # plot color-legends
281
282 for i in range(1, t.size):
283     if QPSK_bits(t[i]) != prevBits or i == t.size - 1:
284         ax3.plot(t[lastPlotted:i], vQPSK(t[lastPlotted:i]), /
285                 c=colors[prevBits])
286     prevBits = QPSK_bits(t[i])
287     lastPlotted = i
288 # we plot different colors for each gray code
289
290 plt.ylabel("Amplitude(V)")
291 plt.xlabel("Time(sec)")
292 plt.grid() # use the grid
293 ax3.set_xlim(left = 0, right = bits*Tb)
294 ax3.set_xticks(tb)
295 ax3.legend(loc = 'lower right')
296 plt.setp(ax3.get_xticklabels(), rotation=90)
297 # we rotate the axis' ticks in order to avoid overlapping

```

8PSK:

Symbol Gray Mapping	
000	s_0
001	s_1
011	s_2
010	s_3
110	s_4
111	s_5
101	s_6
100	s_7

Κατά τη διαμόρφωση 8PSK (8 Phase Shift Keying) κάθε σύμβολο αναπαρίσταται από 3 bit. Έχουμε τις εξής κυματομορφές για την αναπαράσταση των δυαδικών ψηφίων:

$$s_0(t): \text{Acos}(2\pi f_c t), \text{ για bits } 000$$

$$s_1(t): \text{Asin}(2\pi f_c t - \pi/4), \text{ για bit } 001$$

$$s_2(t): -\text{Acos}(2\pi f_c t - \pi/2), \text{ για bits } 011$$

$$s_3(t): -\text{Asin}(2\pi f_c t - 3\pi/4), \text{ για bit } 010$$

$$s_4(t): \text{Acos}(2\pi f_c t - \pi), \text{ για bits } 110$$

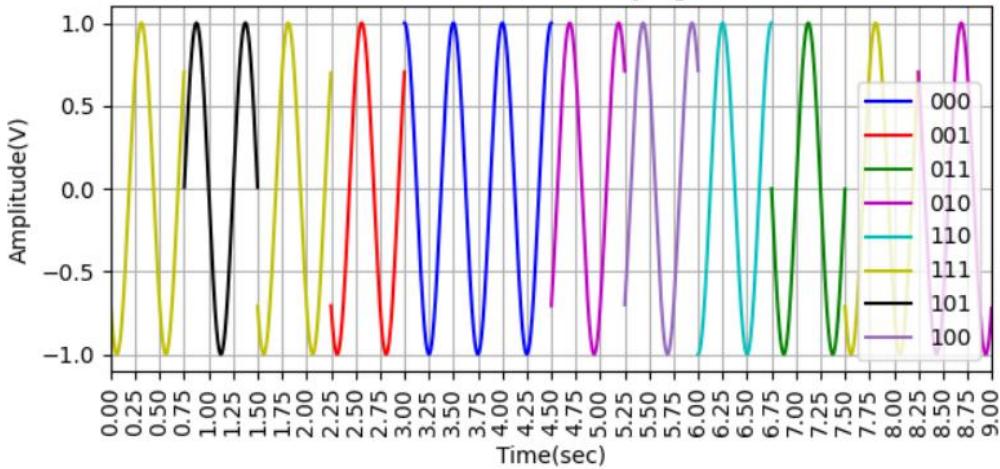
$$s_5(t): \text{Asin}(2\pi f_c t - 5\pi/4), \text{ για bit } 111$$

$$s_6(t): -\text{Acos}(2\pi f_c t - 3\pi/2), \text{ για bits } 101$$

$$s_7(t): -\text{Asin}(2\pi f_c t - 7\pi/4), \text{ για bit } 100$$

$$(\text{για } 0 < t < 3T_b)$$

8 Phase-Shift Keying



```

303 # use / to make spread among many Lines
304 def PSK8_bits(t): # return current bit
305     return str(bitSeq[3*int(t/(3*Tb))]) + \
306     str(bitSeq[3*int(t/(3*Tb))+1]) + \
307     str(bitSeq[3*int(t/(3*Tb))+2])
308
309 def PSK8(t): # which formula to use
310     phase = {gray(i):i*pi/4 for i in range(8)}
311     return A*math.cos(2*pi*fct - phase[PSK8_bits(t)])
312
313 vPSK8 = np.vectorize(PSK8)
314 # make the QPSK() function an element-wise one
315
316 ax4 = fig.add_subplot(224) # create one subplot
317 plt.title("8 Phase-Shift Keying")
318 prevBits = PSK8_bits(t[0])
319 lastPlotted = 0
320 colors = {'000':'b', '001':'r', '011':'g', '010':'m', \
321 '110':'c', '111':'y', '101':'k', '100':'tab:purple'}

```

Υλοποίηση σε Python

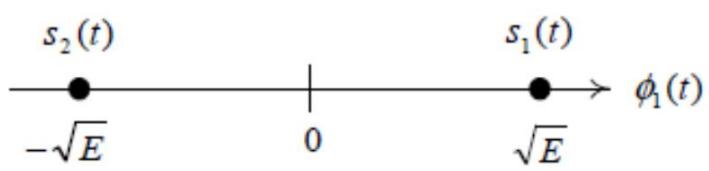
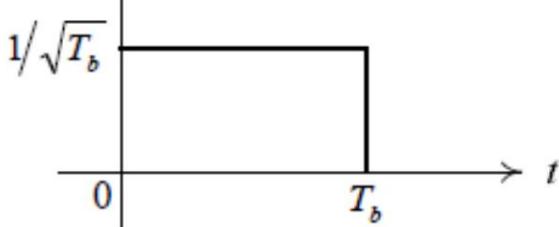
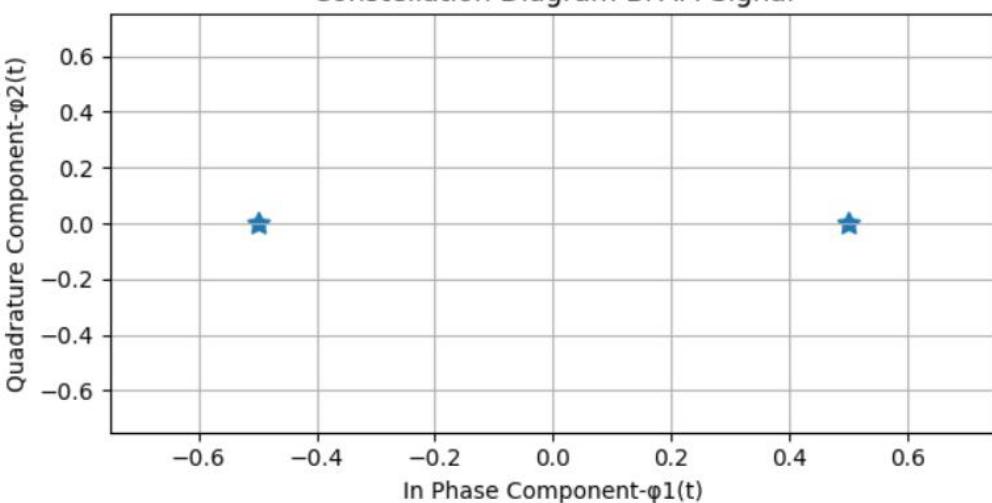
```

321 for i in colors:
322     ax4.plot([],[], c=colors[i], label=i)
323 # plot color-legends
324
325 for i in range(1, t.size):
326     if PSK8_bits(t[i]) != prevBits or i == t.size - 1:
327         ax4.plot(t[lastPlotted:i], vPSK8(t[lastPlotted:i]), \
328         c=colors[prevBits])
329         prevBits = PSK8_bits(t[i])
330         lastPlotted = i
331 # we plot different colors for each gray code
332
333 plt.ylabel("Amplitude(V)")
334 plt.xlabel("Time(sec)")
335 plt.grid() # use the grid
336 ax4.set_xlim(left = 0, right = bits*Tb)
337 ax4.set_xticks(tb)
338 ax4.legend(loc = 'lower right')
339 plt.setp(ax4.get_xticklabels(), rotation=90)
340 # we rotate the axis' ticks in order to avoid overlapping
341 fig.tight_layout() # adjust the spacing between subplots in

```

(γ): Παρουσιάζουμε το διάγραμμα αστερισμού του σήματος BPAM σύμφωνα με τη θεωρία των μαθήματος:

Constellation Diagram-BPAM Signal



```

345 A = 1 # A = 1V
346 phaseBPAM = [0, pi] # The signals' phase
347 eBPAM = A*A*Tb # Energy of bit in BPAM modulation
348 Q = [math.sqrt(eBPAM)*math.sin(phaseBPAM[i]) for i in range(2)]
349 # Quadrature Component
350 I = [math.sqrt(eBPAM)*math.cos(phaseBPAM[i]) for i in range(2)]
351 # In Phase Component
352 figCon = plt.figure(11) # create a new figure
353 ax = figCon.add_subplot(221) # create one subplot
354 plt.title("Constellation Diagram-BPAM Signal")
355 ax.scatter(I, Q, marker=(5, 1), s = 100)
356 plt.ylabel("Quadrature Component- $\phi_2(t)$ ")
357 plt.xlabel("In Phase Component- $\phi_1(t)$ ")
358 plt.grid() # use the grid
359 ax.set_xlim(left = -0.75, right = +0.75)
360 ax.set_ylim(bottom = -0.75, top = +0.75)

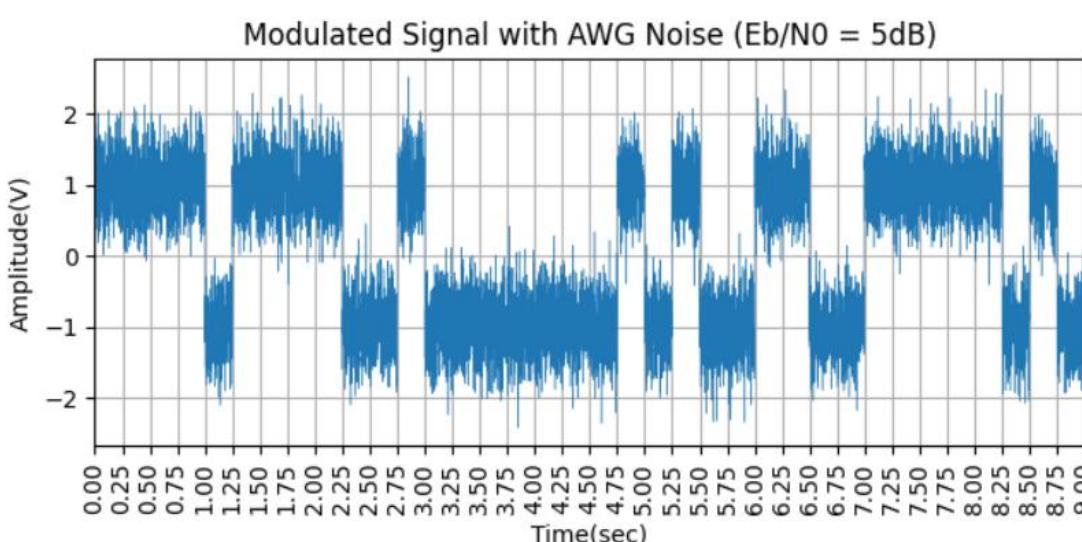
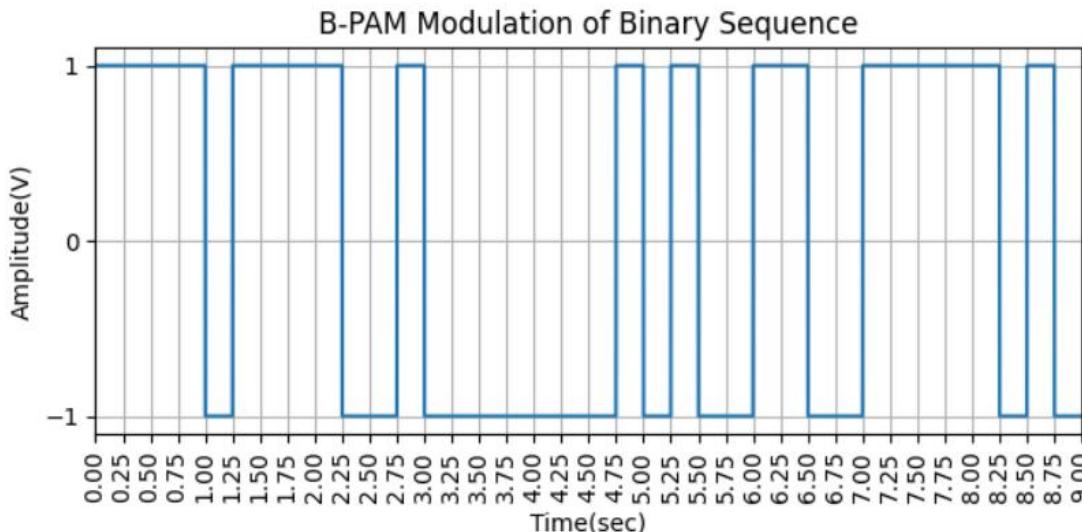
```

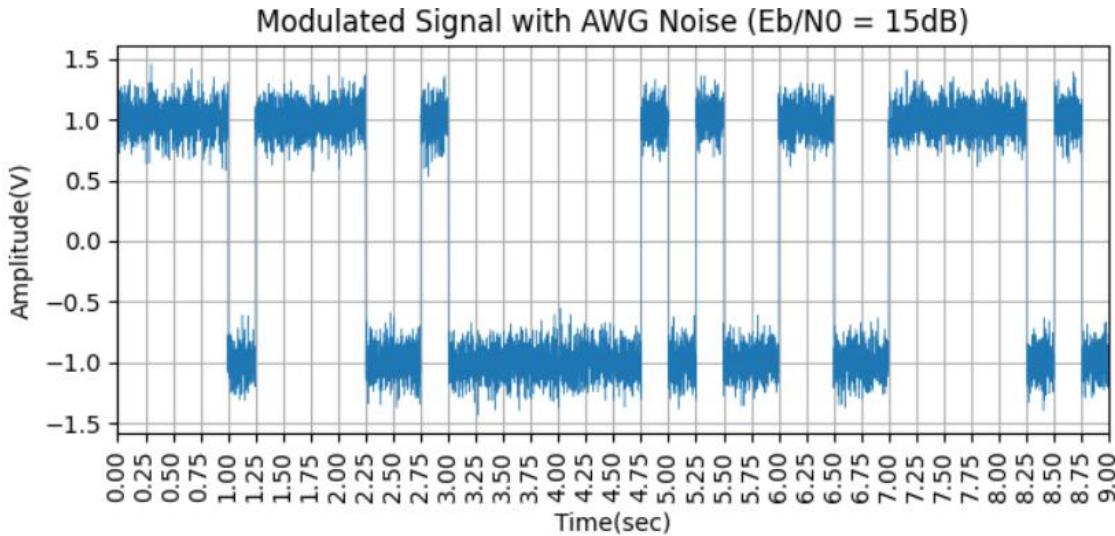
Υλοποίηση σε Python

Σημείωση:

Για τα παρακάτω ερωτήματα προσομοιώνουμε τον θόρυβο ως μιγαδική τυχαία μεταβλητή $Z = X + jY$, όπου οι πραγματικές τυχαίες μεταβλητές X και Y είναι ανεξάρτητες και καθεμία εξ' αυτών ακολουθεί κανονική κατανομή με μηδενική μέση τιμή και τυπική απόκλιση τέτοια ώστε η μονόπλευρη φασματική πυκνότητα ισχύος του θορύβου να είναι $No/2$. Δηλαδή έχουμε δίπλευρη φασματική πυκνότητα ισχύος του θορύβου $No/4$.

(δ'): Προσθέτουμε θόρυβο AWGN στο BPAM σήμα για $E_b/N_0 = 5$ dB και $E_b/N_0 = 15$ dB και έχουμε:





Παρατηρούμε ότι για μικρότερες τιμές (σε dB) του λόγου E_b/N₀ το σήμα παρουσιάζει μεγαλύτερη παραμόρφωση λόγω του προστιθέμενου θορύβου. Αν η παραμόρφωση είναι αρκετά μεγάλη, ο δέκτης δεν μπορεί με ασφάλεια να αποφανθεί την τιμή του bit που εστάλη. Τότε έχουμε σφάλμα στον δέκτη. Στον κώδικα που παρατίθεται παρακάτω φαίνονται με σαφήνεια οι τύποι που χρησιμοποιήθηκαν για την προσθήκη του θορύβου αλλά και την σχεδίαση των διαγραμμάτων.

Προσοχή πρέπει να δοθεί στους τύπους που εμπλέκουν την φασματική πυκνότητα ισχύος του θορύβου. Στις διαφάνειες του μαθήματος έχουμε υποθέσει ότι η μονόπλευρη φασματική πυκνότητα ισχύος του θορύβου είναι N₀, ενώ τώρα σύμφωνα με την εκφώνηση, έχουμε μονόπλευρη φασματική πυκνότητα ισχύος του θορύβου είναι N₀/2. Το γεγονός αυτό πρέπει να ληφθεί υπόψη στους χρησιμοποιούμενους τύπους (όπου ratio(given) = $\frac{E_b}{N_0}$, ratio'(used in SNR calculations) = $\frac{E_b}{\frac{N_0}{2}} = 2 \cdot \text{ratio}$).

```

377 M = 2 # We have B-PAM
378 k = math.log(M, 2)
379 ratioIndB = np.array([5, 15]) # ratio(dB) = 10log(Eb/(N0/2))
380 ratio = 10**(ratioIndB/10)
381 SNR = 2*k*ratio
382 noisePower = stdDev(t.size, vBPAM(t))/SNR
383 noiseMean = 0
384 for i in range(2):
385     noiseX = np.random.normal(noiseMean, noisePower[i]**(1/2.0), t.size)
386     noiseY = np.random.normal(noiseMean, noisePower[i]**(1/2.0), t.size)
387     sigWnoise = noiseX + vBPAM(t)
388     ax1 = fig.add_subplot(2,2,i+2) # create one subplot
389     ax1.set_title("Modulated Signal with AWG Noise (E_b/N_0 = {}dB)".format(ratioIndB[i]))
390     ax1.set_ylabel("Amplitude(V)")
391     ax1.set_xlabel("Time(sec)")
392     ax1.plot(t, sigWnoise, linewidth=0.3)
393     ax1.grid() # use the grid

```

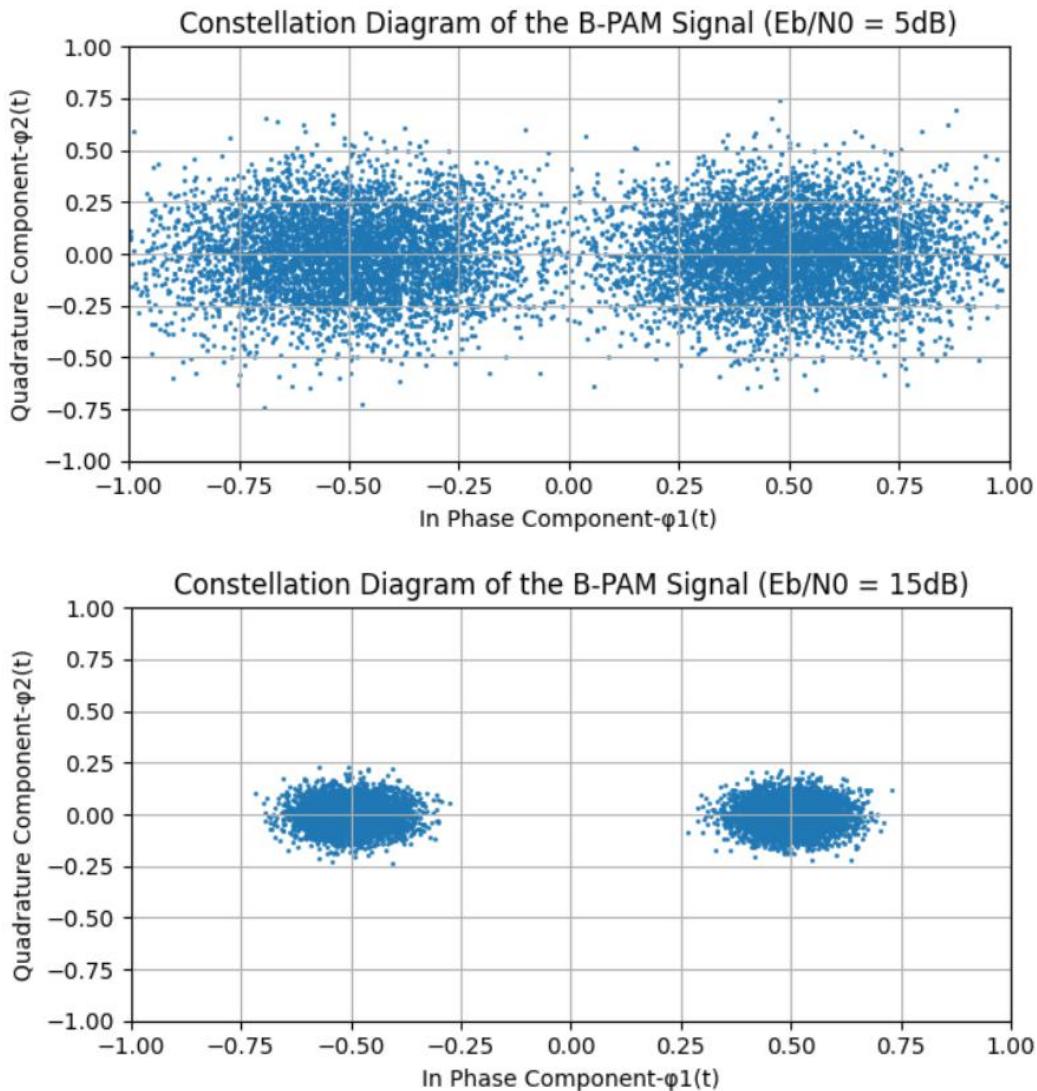
```

395     ax1.set_xlim(left = 0, right = bits*Tb)
396     ax1.set_xticks(tb)
397     plt.setp(ax1.get_xticklabels(), rotation=90)
398     # Question (e')
399     ax2 = figCon.add_subplot(2, 2, 2+i) # create one subplot
400     ax2.set_title("Constellation Diagram of the B-PAM Signal (E_b/N_0 = {}dB)".format(ratioIndB[i]))
401     ax2.set_xlabel("Quadrature Component-φ2(t)")
402     ax2.set_ylabel("In Phase Component-φ1(t)")
403     ax2.scatter(sigWnoise*math.sqrt(eBPAM), noiseY*math.sqrt(eBPAM), s = 1)
404     ax2.grid() # use the grid
405     ax2.set_xlim(left = -1, right = +1)
406     ax2.set_ylim(bottom = -1, top = +1)
407
408 figCon.tight_layout()
409 fig.tight_layout() # adjust the spacing between subplots in
410 # order to avoid text overlapping

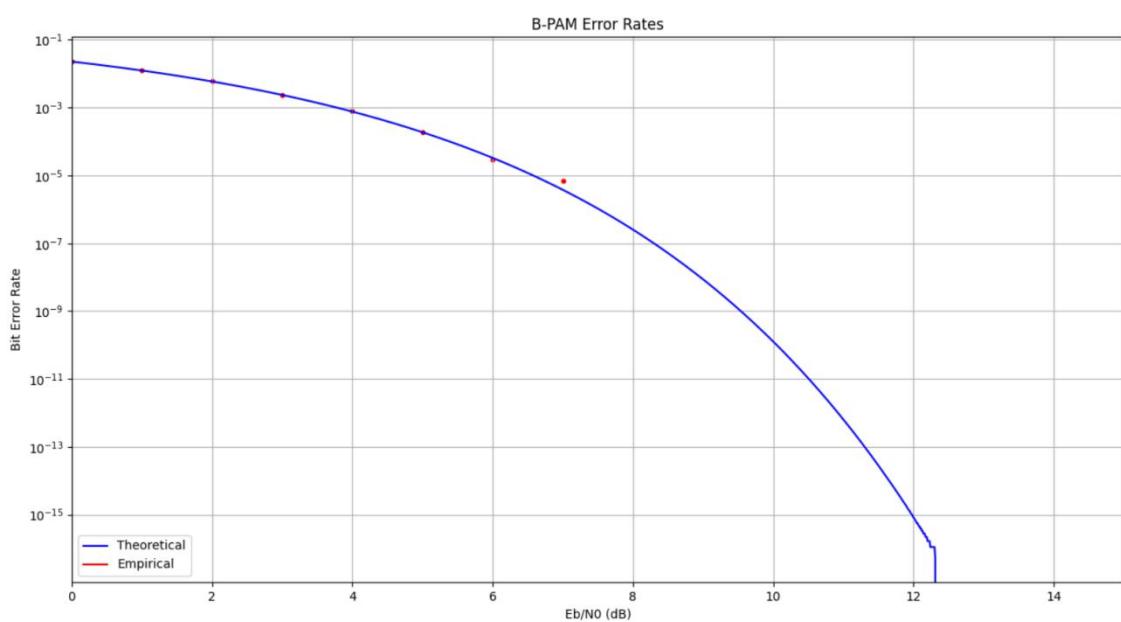
```

Υλοποίηση σε Python

(ε'): Παρουσιάζουμε τα διαγράμματα αστερισμών για τα σήματα που προέκυψαν στο προηγούμενο ερώτημα. Η υλοποίηση σε Python φαίνεται ακριβώς από πάνω. Έχει γίνει κανονικοποίηση και του κατακόρυφου άξονα με την τετραγωνική ρίζα της ενέργειας του bit σε διαμόρφωση BPAM. Παρατηρούμε ότι η προσθήκη θορύβου εκτρέπει το διάνυσμα του αρχικού σήματος στο I-Q επίπεδο, τόσο στην συμφασική όσο και στην ορθογωνική συνιστώσα. Κατά αυτόν τον τρόπο τα διαγράμματα αστερισμών μας επιτρέπουν να παρουσιάζουμε συνοπτικά την επίπτωση που έχει ο θόρυβος στο αρχικό σήμα και να αντιληφθούμε διαισθητικά την πιθανότητα σφάλματος στον δέκτη.



(ζ): Παρουσιάζουμε το διάγραμμα πιθανότητας εσφαλμένου ψηφίου (BER) συναρτήσει του λόγου E_b/N_0 :



Η συνεχής γραμμή αναπαριστά τη θεωρητική πιθανότητα εσφαλμένου ψηφίου για κάθε τιμή του E_b/N_0 για διαμόρφωση BPAM με βέλτιστο δέκτη. Συνεπώς για να υπολογίσουμε την εμπειρική συχνότητα εμφάνισης σφάλματος στο σήμα μας (υπό την επίδραση θορύβου) χρησιμοποιούμε προσαρμοσμένο φίλτρο και ανιχνευτή σφάλματος. Η ακρίβεια της μέτρησής μας περιορίζεται από την υπολογιστική μας

ισχύ, καθώς μετά από ένα όριο bit και τμηματοποίησης του Tb το πρόγραμμα αργεί υπερβολικά.

Συνεπώς για να αναπαραστήσουμε με μεγαλύτερη ακρίβεια την συχνότητα σφάλματος πρέπει να αυξήσουμε τόσο των αριθμό των bits στην δυαδική ακολουθία όσο και την τμηματοποίηση του χρόνου (ώστε να προσεγγιστεί η συνεχής φύση του). Για τις εμπειρικές μετρήσεις (σημεία με κόκκινο χρώμα στο διάγραμμα) αρκεστήκαμε σε 10^6 bits και σε $Tb/2$ τμηματοποίηση χρόνου.

Παρατηρούμε ότι η προσέγγιση των εμπειρικών μετρήσεων είναι ικανοποιητική.

Σημείωση: Από μια τιμή και μετά του λόγου E_b/N_0 η συχνότητα σφάλματος στις εμπειρικές μετρήσεις υπολογίζεται μηδενική, εξ' ού και η απουσία των σημείων αντών στο λογαριθμικό διάγραμμα.

```

433 # calculate the empirical probability error
434 bits = 10**6
435 bitSeq = np.random.randint(2, size = bits) # binary sequence
436 divs = 2 # how many division of time in Tb
437 t = np.linspace(0, bits*Tb, divs*bits, endpoint=False) # time
438 ratioIndB = np.arange(16) # ratio values for plotting
439 for i in range(ratioIndB.size):
440     ratio = 10***(ratioIndB[i]/10)
441     noisePower = stdDev(t.size, vBPAM(t))/(2*ratio)
442     # we must multiply with 2 because we have N0/2 one sided
443     # spectral density for the noise, so N0/4 two sided
444     noise = np.random.normal(0, noisePower***(1/2.0), t.size)
445     sigWnoise = noise + vBPAM(t) # add noise to our signal
446     sums = np.zeros(bits)
447     for time in range(t.size): # using matched filter
448         sums[int(t[time]/Tb)] += sigWnoise[time]
449
450     errors = 0 # count the errors
451     for bit in range(bits): # step = i*Tb
452         if sums[bit]*BPAM(bit*Tb)<0:
453             errors += 1
454         elif sums[bit] == 0 and np.random.randint(2, size = 1) == 1:
455             errors += 1
456     errorFreq = errors / bits # calculate error frequency
457     ax.scatter(ratioIndB[i], errorFreq, c='r', s=10)
458
459 ax.plot([],[], c='r', label='Empirical')
460 ax.legend(loc = 'lower left')
461 plt.show() # show the figures and their plots

```

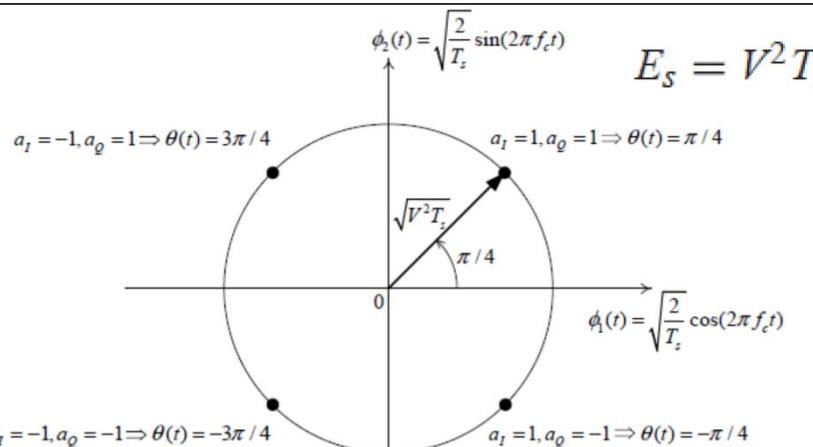
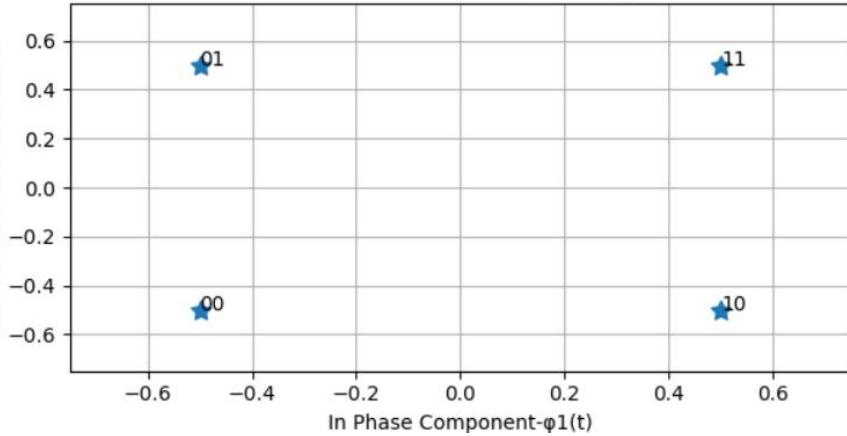
Υλοποίηση σε Python

4^o Ερώτημα:

Διαμορφώνουμε την δυαδική ακολουθία του προηγούμενου ερωτήματος κατά QPSK (βασικής ζώνης και όχι πάνω σε φέρον σήμα).

(α'): Παρουσιάζουμε το διάγραμμα αστερισμού θεωρώντας απεικόνιση με κωδικοποίηση ($\pi/4$) Gray:

Constellation Diagram-QPSK Signal



Υλοποίηση σε Python

```

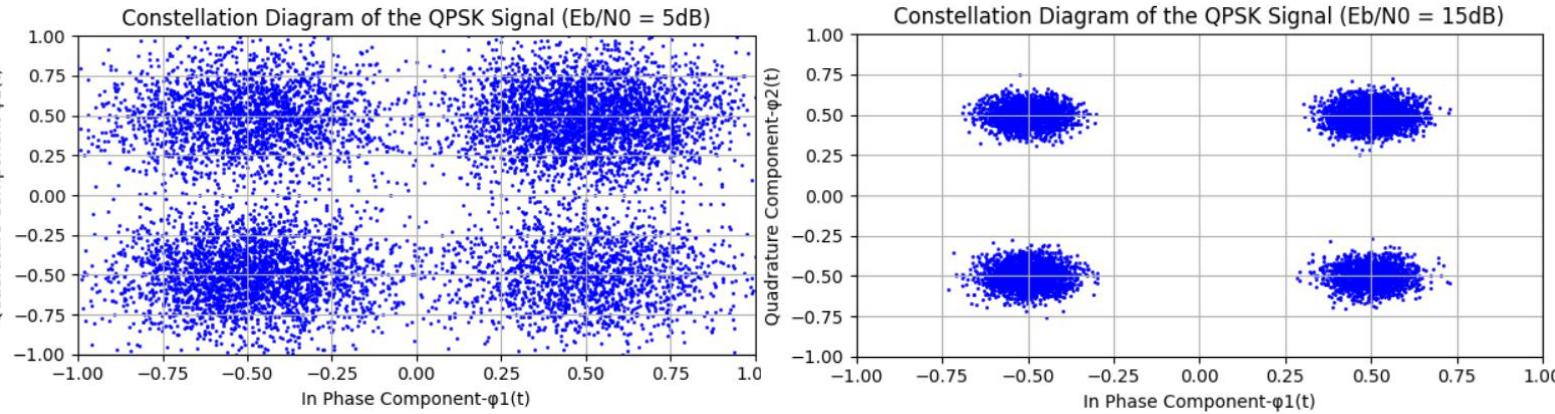
466 bits = 36
467 bitSeq = np.random.randint(2, size = bits)
468 # generate random 36-bits sequence
469 # discrete uniform distribution
470 print(bitSeq) # print the sequence
471 Tb = 0.25 # bit duration (sec), Tsymbol = 2Tb
472 A = 1 # Amplitude (V)
473 eQPSK = A**2*Tb # Bit Energy for QPSK, SymbolEnergy = 2*eQPSK
474 phaseQPSK = [-3*pi/4, 3*pi/4, -pi/4, pi/4] # The signals' phase
475 grayCode = ['00', '01', '10', '11'] # Gray coded signals
476 Q = [math.sqrt(2*eQPSK)*math.sin(phaseQPSK[i]) for i in range(4)]
477 # Quadrature Component
478 I = [math.sqrt(2*eQPSK)*math.cos(phaseQPSK[i]) for i in range(4)]
479 # In Phase Component
480 figCon = plt.figure(14) # create a new figure
481 ax = figCon.add_subplot(221) # create one subplot
482 plt.title("Constellation Diagram-QPSK Signal")
483 ax.scatter(I, Q, marker=(5, 1), s = 100)
484 for i in range(4):
485     ax.annotate(grayCode[i], (I[i],Q[i]), fontsize=10)
486 plt.ylabel("Quadrature Component-phi2(t)")
487 plt.xlabel("In Phase Component-phi1(t)")
488 plt.grid() # use the grid
489 ax.set_xlim(left = -0.75, right = +0.75)
490 ax.set_ylim(bottom = -0.75, top = +0.75)

```

$$\theta(t) = \begin{cases} \pi/4, & a_I = +1, a_Q = +1 \text{ (bits 11)} \\ -\pi/4, & a_I = +1, a_Q = -1 \text{ (bits 10)} \\ 3\pi/4, & a_I = -1, a_Q = +1 \text{ (bits 01)} \\ -3\pi/4, & a_I = -1, a_Q = -1 \text{ (bits 00)} \end{cases}$$

Θεωρία διαγράμματος αστερισμού QPSK:

(β'): Παρουσιάζουμε το διάγραμμα αστερισμού του αρχικού σήματος, έχοντας προσθέσει σε αυτό θόρυβο AWGN με $E_b/N_0 = 5$ dB και $E_b/N_0 = 15$ dB:



```

494 t = np.linspace(0, bits*Tb, 10000, endpoint = False)
495 # consider continuous t, don't include t = bits*Tb
496 grayMapped = {'00':-3*pi/4, '01':3*pi/4, '10':-pi/4, '11':pi/4}
497 M = 4 # We have QPSK
498 k = math.log(M, 2)
499 ratioIndB = np.array([5, 15]) # ratio(dB) = 10Log(Eb/(N0/2))
500 ratio = 10***(ratioIndB/10)
501 SNR = 2*k*ratio
502 def QPSK_bits(t):
503     return str(bitSeq[2*int(t/(2*Tb))])+str(bitSeq[2*int(t/(2*Tb))+1])
504
505 noisePower = eQPSK/Tb/SNR # power of given QPSK signal = eQPSK/Tb
506 noiseMean = 0

```

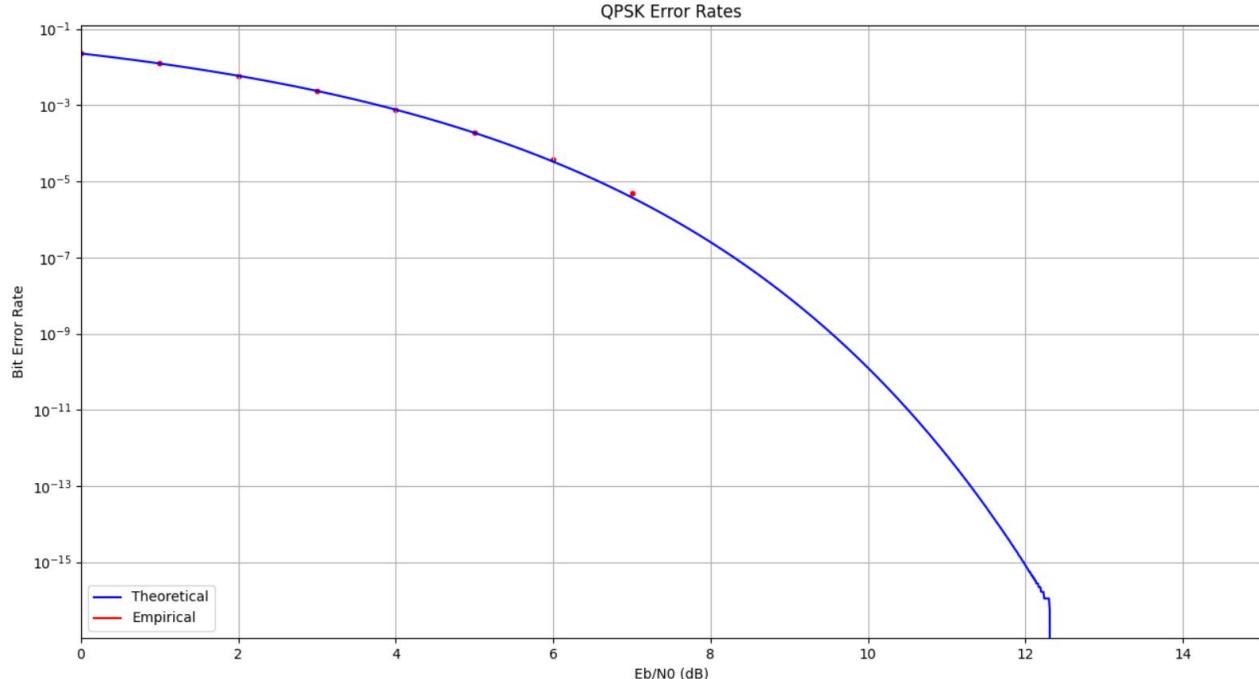
```

507 for i in range(2):
508     noiseX = np.random.normal(noiseMean, noisePower[i]**(1/2.0), t.size)
509     noiseY = np.random.normal(noiseMean, noisePower[i]**(1/2.0), t.size)
510 ax2 = figCon.add_subplot(2, 2, 2*i) # create one subplot
511 ax2.set_title("Constellation Diagram of the QPSK Signal (Eb/N0 = {}dB)".format(ratioIndB[i]))
512 ax2.set_ylabel("Quadrature Component-φ2(t)")
513 ax2.set_xlabel("In Phase Component-φ1(t)")
514 ax2.grid()
515 ax2.set_xlim(left = -1, right = +1)
516 ax2.set_ylim(bottom = -1, top = +1)
517 for j in range(t.size):
518     sigWnoiseX = noiseX[j] + math.cos(grayMapped[QPSK_bits(t[j])])
519     sigWnoiseY = noiseY[j] + math.sin(grayMapped[QPSK_bits(t[j])])
520     ax2.scatter(sigWnoiseX*math.sqrt(2*eQPSK), sigWnoiseY*math.sqrt(2*eQPSK), s = 1, c='b')
521 figCon.tight_layout()

```

Υλοποίηση σε Python

(γ'): Παρουσιάζουμε το διάγραμμα πιθανότητας εσφαλμένου ψηφίου (BER) συναρτήσει του λόγου E_b/N_0 :



Η συνεχής γραμμή αναπαριστά τη θεωρητική πιθανότητα εσφαλμένου ψηφίου για κάθε τιμή του E_b/N_0 για διαμόρφωση QPSK με βέλτιστο δέκτη. Συνεπώς για να υπολογίσουμε την εμπειρική συχνότητα εμφάνισης σφάλματος στο σήμα μας (υπό την επίδραση θορύβου) χρησιμοποιούμε προσαρμοσμένο φίλτρο και ανιχνευτή σφάλματος. Η ακρίβεια της μέτρησής μας περιορίζεται από την υπολογιστική μας ισχύ καθώς μετά από ένα όριο bit και τμηματοποίησης του Tb το πρόγραμμα αργεί υπερβολικά.

Συνεπώς για να αναπαραστήσουμε με μεγαλύτερη ακρίβεια την συχνότητα σφάλματος πρέπει να αυξήσουμε τόσο των αριθμό των bits στην δυαδική ακολουθία όσο και την τμηματοποίηση του χρόνου (ώστε να προσεγγιστεί η συνεχής φύση του). Για τις εμπειρικές μας μετρήσεις (σημεία με κόκκινο χρώμα στο διάγραμμα) αρκεστήκαμε σε 10^6 bits και σε Tb τμηματοποίηση χρόνου.

Παρατηρούμε ότι η προσέγγιση των εμπειρικών μετρήσεων είναι ικανοποιητική.

Σημείωση: Από μια τιμή και μετά του λόγου E_b/N_0 η συχνότητα σφάλματος στις εμπειρικές μετρήσεις υπολογίζεται μηδενική, εξ ού και η απουσία των σημείων αντών στο λογαριθμικό διάγραμμα.

Παρατηρούμε ότι η πιθανότητα σφάλματος για διαμόρφωση BPAM και QPSK είναι ίδια.

Ωστόσο για ένα δεδομένο εύρος ζώνης συχνοτήτων μετάδοσης, ένα σύστημα QPSK φέρει διπλάσιο αριθμό bit πληροφορίας σε σχέση με το αντίστοιχο δυαδικό σύστημα PSK.

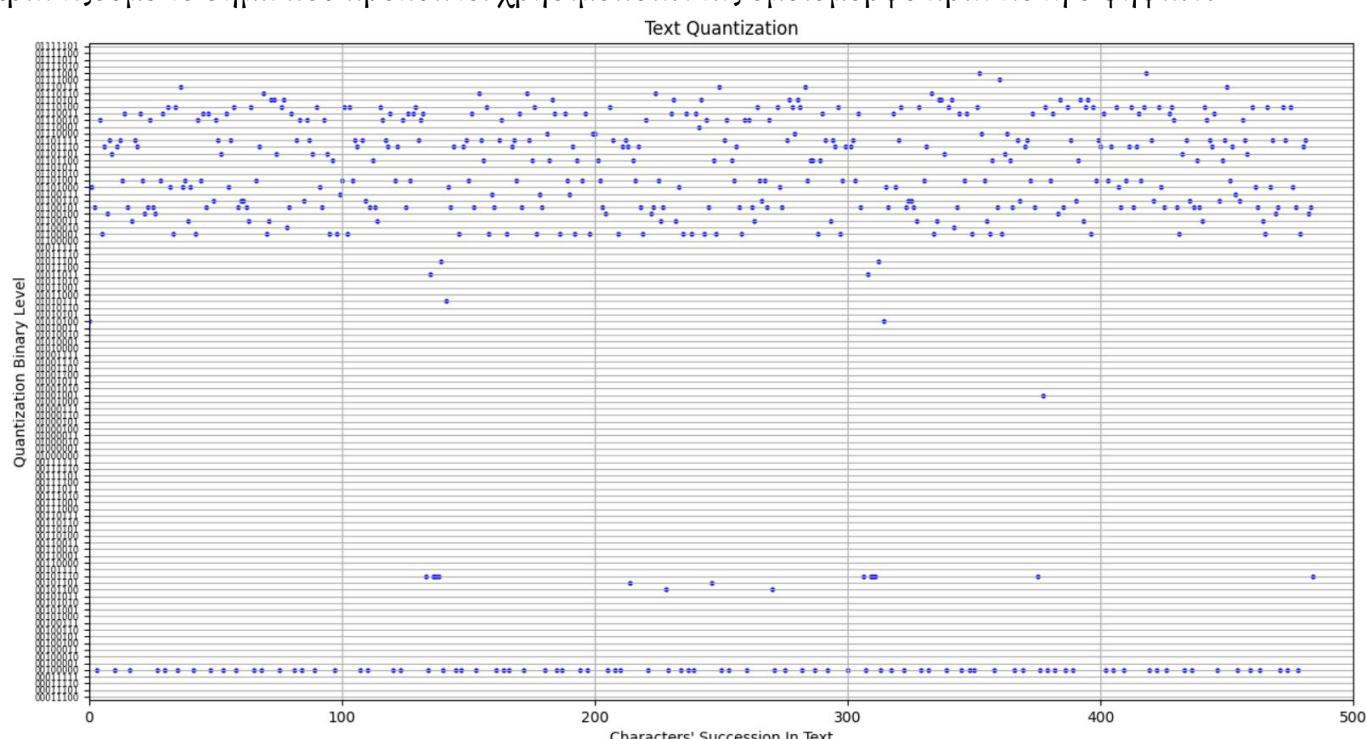
```
524 def QPSK_bitsRecon(x, y):
525     if x == 0:
526         x = 2*np.random.randint(2, size=1)-1
527     if y == 0:
528         y = 2*np.random.randint(2, size=1)-1
529 # if x == 0 or y == 0 we consider them positive
530 # or negative randomly
531     if x < 0 and y < 0:
532         return '00'
533     elif x < 0 and y > 0:
534         return '01'
535     elif x > 0 and y > 0:
536         return '11'
537     elif x > 0 and y < 0:
538         return '10'
539
540 fig = plt.figure(15) # create a new figure
541 ax = fig.add_subplot(111) # create one subplot
542 plt.title("QPSK Error Rates")
543 plt.ylabel("Bit Error Rate")
544 plt.xlabel("Eb/N0 (dB)")
545 plt.grid() # use the grid
546 ax.set_xlim(left = 0, right = 15)
547 ax.set_yscale('log')
548
549 # calculate the theoretical probability error
550 ratioIndB = np.arange(0, 16, 0.01)
551 ratio = 10***(ratioIndB/10)
552 Perror = vQ((2*2*ratio)**(1/2.0)) # Probability error in BPAM
553 ax.plot(ratioIndB, Perror, c='b', label='Theoretical')
554
555 import binascii
556 f = open("rice_odd.txt", "r") # el19145
557 # 1 + 4 + 5 = 10, 1 + 0 = 1 (odd)
558 binary = bin(int.from_bytes(f.read().encode(), 'big'))
559 binary = binary[0] + binary[2:]
560 # remove b character which denotes binary number in python
```

```
556     bits = 10**6
557     bitSeq = np.random.randint(2, size = bits) # binary sequence
558     divs = 1 # how many division of time in Tb
559     t = np.linspace(0, bits*Tb, divs*bits, endpoint=False) # time
560     ratioIndB = np.arange(16) # ratio values for plotting
561     for i in range(ratioIndB.size):
562         ratio = 10***(ratioIndB[i]/10)
563         noisePower = eQPSK/Tb/(2*k*ratio)
564         # we must multiply with 2 because we have N0/2 one sided
565         # spectral density for the noise, so N0/4 two sided
566         noiseX = np.random.normal(noiseMean, noisePower***(1/2.0), t.size)
567         noiseY = np.random.normal(noiseMean, noisePower***(1/2.0), t.size)
568         Xs = np.zeros(int(bits/2))
569         Ys = np.zeros(int(bits/2))
570         for time in range(t.size): # using matched filter
571             Xs[int(t[time]/(2*Tb))] += noiseX[time] +
572             math.cos(grayMapped[QPSK_bits(t[time])])
573             Ys[int(t[time]/(2*Tb))] += noiseY[time] +
574             math.sin(grayMapped[QPSK_bits(t[time])])
575         errors = 0
576         for j in range(Xs.size):
577             if str(bitSeq[2*j]) + /
578                 str(bitSeq[2*j+1]) != QPSK_bitsRecon(Xs[j], Ys[j]):
579                 errors += 1
580         errorFreq = errors / bits # calculate error frequency
581         ax.scatter(ratioIndB[i], errorFreq, c='r', s=10)
582
583 ax.plot([],[], c='r', label='Empirical')
584 ax.legend(loc = 'lower left')
```

Υλοποίηση σε Python

(δ', ii) Διαβάζουμε το δοθέν αρχείο κειμένου και το μετατρέπουμε σε δυαδική κωδικοσειρά ASCII:

(ii) Κβαντίζουμε το σήμα που προκύπτει χρησιμοποιώντας ομοιόμορφο κβαντιστή 8 ψηφίων:



Για λόγους ευκρινούς απεικόνισης στο παραπάνω διάγραμμα, δεν προβάλλονται όλα τα επίπεδα κβαντιστή.

```

608 fig = plt.figure(16) # create a new figure
609 ax = fig.add_subplot(111) # create one subplot
610 plt.title("Text Quantization")
611 plt.ylabel("Quantization Binary Level")
612 plt.xlabel("Characters' Succession In Text")
613 plt.grid() # use the grid
614 plt.yticks([i for i in range(levels)], [(bin(i)[0]+bin(i)[2:]).zfill(8) / 
615 for i in range(levels)])
616 plt.yticks(fontsize=6)
617 for i in range(0, len(binary), 8):
618     ax.scatter(i/8, int(binary[i:i+8], 2), c='b', s=5)
619 ax.set_xlim(left = 0, right = 500)

```

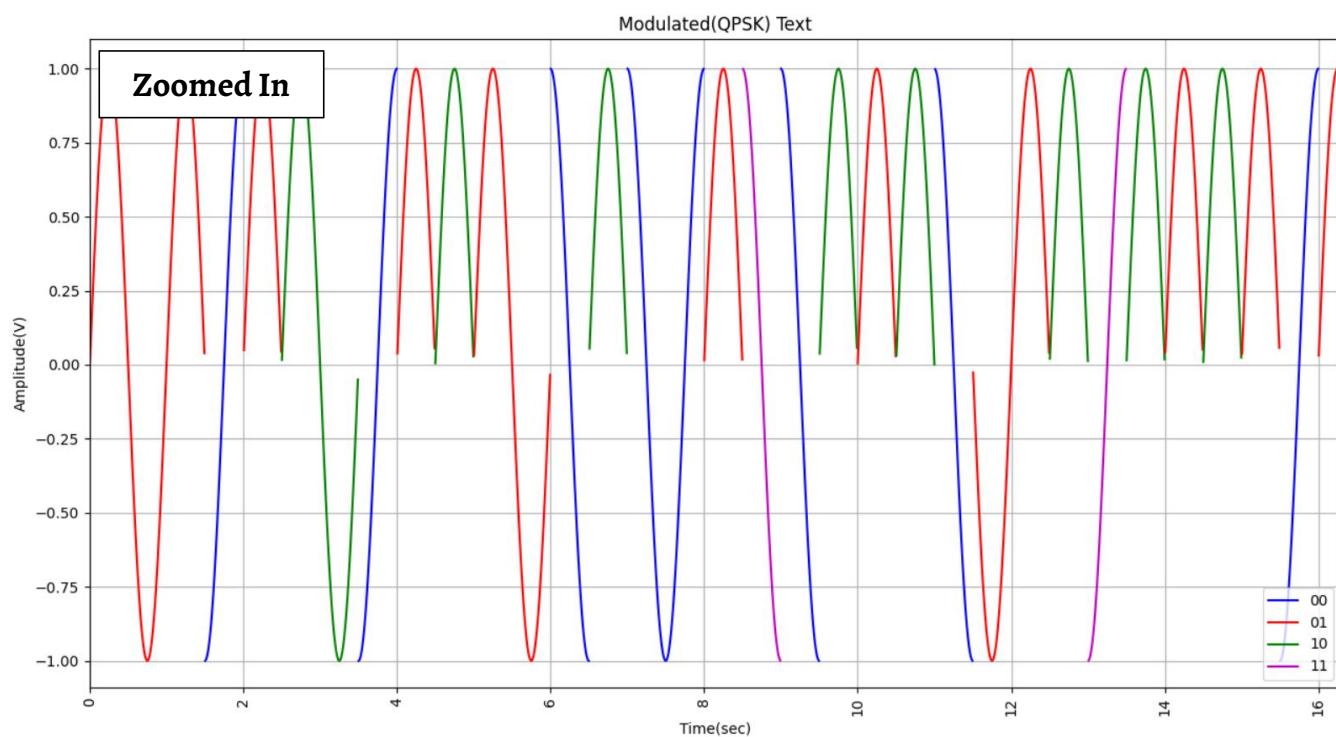
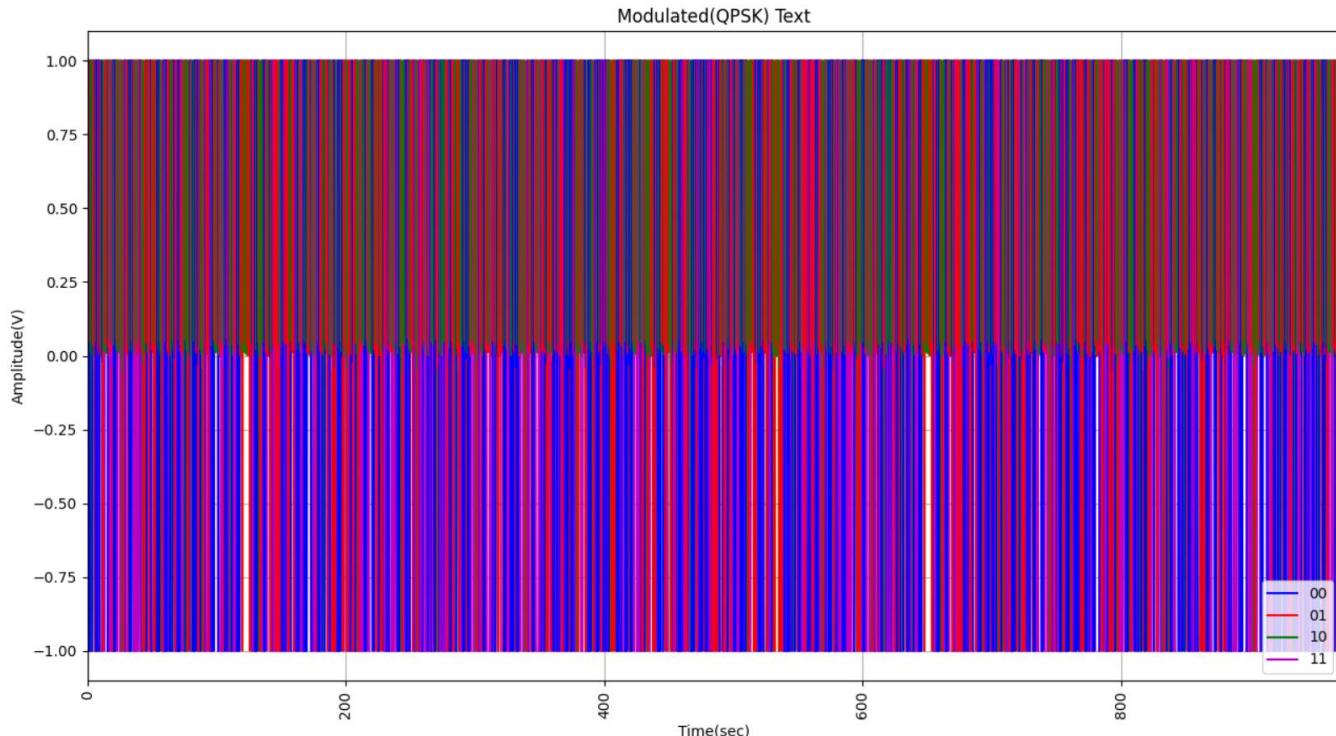
```

597 bits = 8 # fm = 1kHz odd
598 levels = 2**bits # Levels of quantization
599 D = 2**bits/levels # quantization step=valuesRange/levels
600 # the characters (8-digit binary numbers) can take the
601 # discrete values from 00000000 to 11111111 (not Gray coded)
602
603 # So we can infer that the output quantized value
604 # is equal to the input value (8-digit binary number),
605 # as the quantizer and each character uses 8 bits and
606 # the string is already a discrete signal

```

Υλοποίηση σε Python

(iii) Διαμορφώνουμε το κβαντισμένο σήμα χρησιμοποιώντας διαμόρφωση QPSK θεωρώντας απεικόνιση με κωδικοποίηση Gray (όχι κωδικοποίηση $\pi/4$), σύμβολα πλάτους 1V και συχνότητα φέροντος 1Hz:



```

626 Tb = 0.25
627 A = 1
628 def QPSK_bits(t):
629     return str(binary[2*int(t/(2*Tb))])+str(binary[2*int(t/(2*Tb))+1])
630
631 def QPSK(t): # which formula to use (gray coded)
632     if QPSK_bits(t) == '00':
633         return A*math.cos(2*pi*fc*t)
634     elif QPSK_bits(t) == '01':
635         return A*math.sin(2*pi*fc*t)
636     elif QPSK_bits(t) == '11':
637         return -A*math.cos(2*pi*fc*t)
638     elif QPSK_bits(t) == '10':
639         return -A*math.sin(2*pi*fc*t)
640
641 vQPSK = np.vectorize(QPSK)
642 # make the QPSK() function an element-wise one
643
644 t = np.linspace(0, len(binary)*Tb, 100000, endpoint = False)
645 # consider continuous t, don't include t = bits*Tb
646 tb = np.arange(0, len(binary)*Tb+Tb, Tb) # used for ticks

```

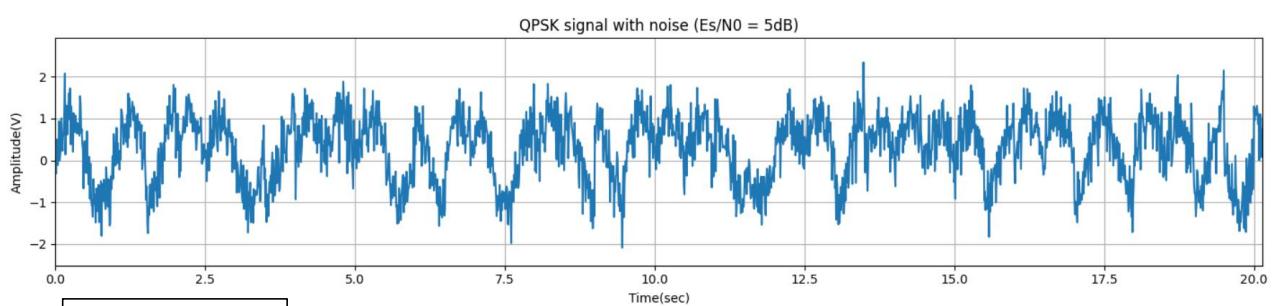
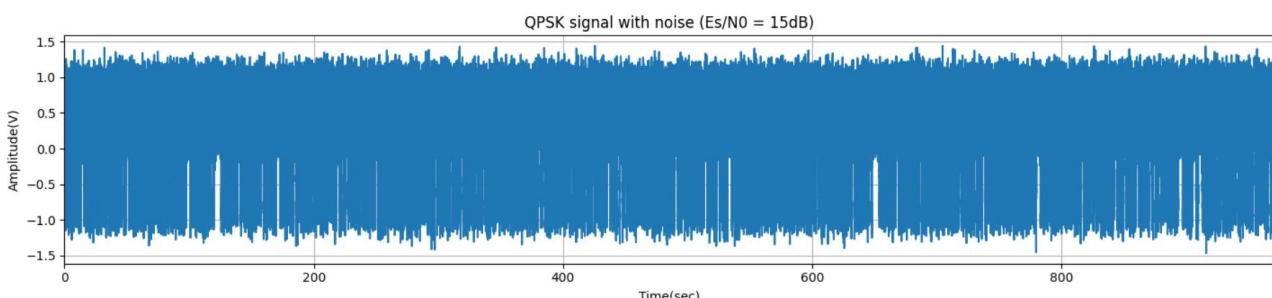
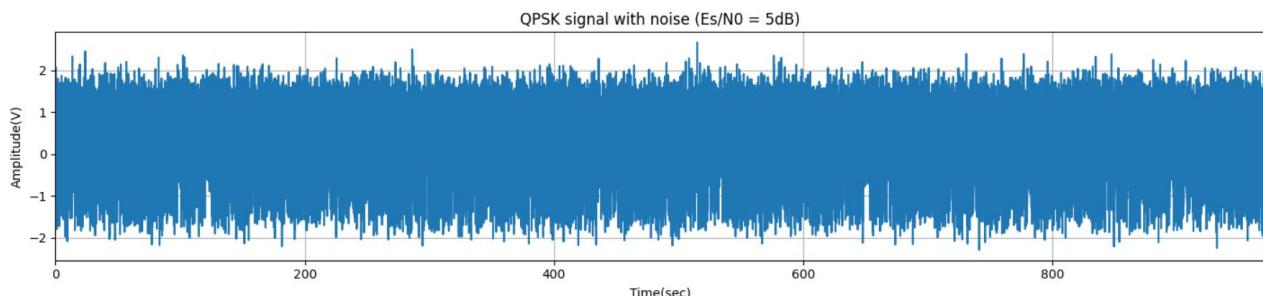
Υλοποίηση σε Python

```

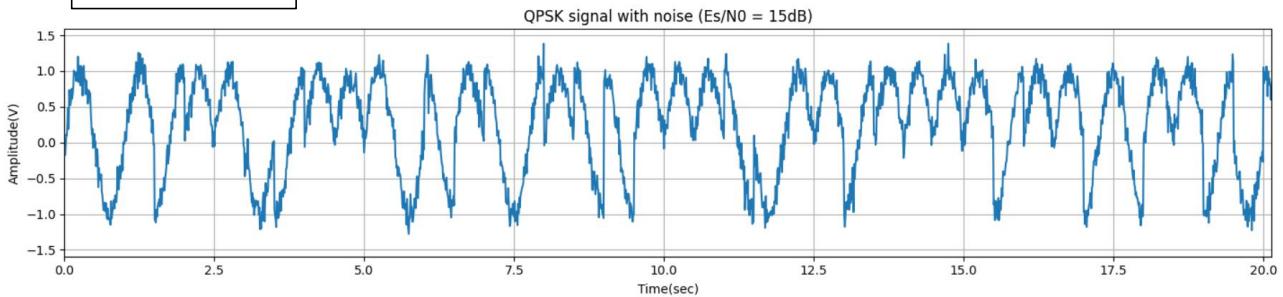
648 fig = plt.figure(17) # create a new figure
649 ax = fig.add_subplot(111) # create one subplot
650 plt.title("Modulated(QPSK) Text")
651 prevBits = QPSK_bits(t[0])
652 lastPlotted = 0
653 colors = {'00':'b', '01':'r','10':'g','11':'m'}
654 for i in colors:
655     ax.plot([],[], c=colors[i], label=i)
656 for i in range(1, t.size):
657     if QPSK_bits(t[i]) != prevBits or i == t.size - 1:
658         ax.plot(t[lastPlotted:i], vQPSK(t[lastPlotted:i]), c=colors[prevBits])
659         prevBits = QPSK_bits(t[i])
660         lastPlotted = i
661
662 # we plot different colors for each gray code
663
664
665 plt.ylabel("Amplitude(V)")
666 plt.xlabel("Time(sec)")
667 plt.grid() # use the grid
668 ax.set_xlim(left = 0, right = len(binary)*Tb)
669 ax.legend(loc = 'lower right')
670 plt.setup(ax.get_xticklabels(), rotation=90)

```

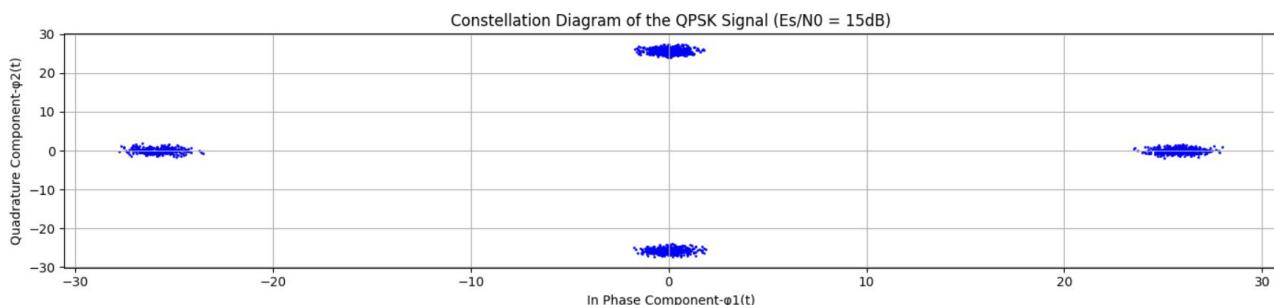
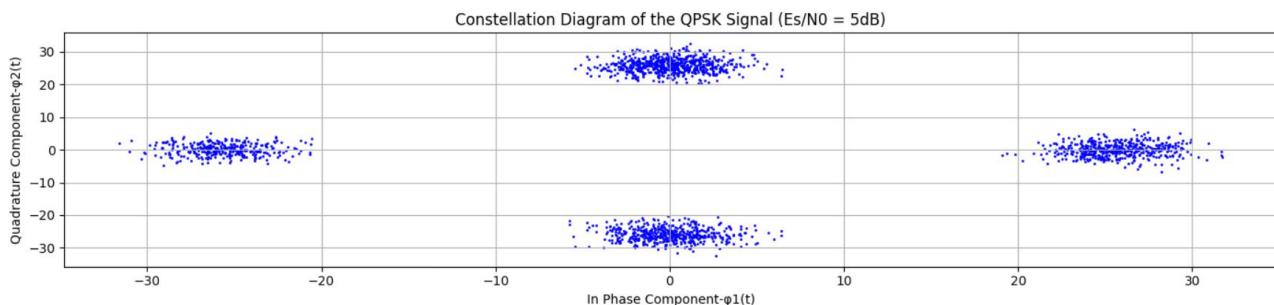
(iv) Παράγουμε θόρυβο AWGN και τον προσθέτουμε στο σήμα QPSK για $E_s/N_0 = 5 \text{ dB}$ και $E_s/N_0 = 15 \text{ dB}$:



Zoomed In



(v) Αποδιαμορφώνουμε και παρουσιάζουμε το διάγραμμα αστερισμών για τα σήματα που προέκυψαν:



Σημείωση: Έχουμε πολλαπλασιάσει κάθε συντεταγμένη με μία σταθερά για να είναι το διάγραμμα ευκρινές.

Ωστόσο η μορφή του διαγράμματος αστερισμού παραμένει αμετάβλητη.

```

667 # (iv) Noise Addition
668 ratioIndB = np.array([5, 15]) # ratio(dB) = 10log(Es/(N0/2))
669 ratio = 10**ratioIndB/10
670 SNR = 2*ratio # we have N0/2 one-sided noise power spectral density
671 eQPSK = A*A*Tb # Energy of bit in QPSK modulation
672 noisePower = eQPSK/Tb/SNR # power of given QPSK signal = eQPSK/Tb
673 noiseMean = 0
674 fig = plt.figure(18) # create a new figure
675 figCon = plt.figure(19) # create a new figure
676 for i in range(2):
677     noise = np.random.normal(noiseMean, noisePower[i]**(1/2.0), t.size)
678     ax = fig.add_subplot(2,1,i+1) # create one subplot
679     ax.set_title("QPSK signal with noise (Es/N0 = {}dB)".format(ratioIndB[i]))
680     ax.set_ylabel("Amplitude(V)")
681     ax.set_xlabel("Time(sec)")
682     ax.grid() # use the grid
683     ax.set_xlim(left = 0, right = len(binary)*Tb)
684     sigWnoise = noise + vQPSK(t) # noise added to original signal
685     ax.plot(t, sigWnoise)

```

(vi) Υπολογίζουμε την πιθανότητα εσφαλμένου ψηφίου BER εμπειρικά και θεωρητικά:

For Es/N0(db)=5
Empirical BER: 0.0
Theoretical BER: 0.00595386714777868

For Es/N0(db)=15
Empirical BER: 0.0
Theoretical BER: 8.881784197001252e-16

```

786 phases = [0, pi/2, pi, 3*pi/2]
787 codes = ['00', '01', '11', '10']
788 sI = [math.cos(k) for k in phases]
789 sQ = [math.sin(k) for k in phases]
790 binaryRecon = ''
791 for bitPair in range(Xs.size):
792     min = 10000 # one big number for min var init
793     minJ = 0
794     for j in range(4): # find the min distance among original sig
795         if (Xs[bitPair]-sI[j])**2 + (Ys[bitPair]-sQ[j])**2 < min:
796             min = (Xs[bitPair]-sI[j])**2 + (Ys[bitPair]-sQ[j])**2
797             minJ = j
798     binaryRecon += codes[minJ] # s_j decided
799
800 errors = 0
801 for bit in range(len(binaryRecon)):
802     if binaryRecon[bit] != binary[bit]:
803         errors += 1
804 BERempirical = errors/len(binaryRecon)
805 BERtheoretical = Q(math.sqrt(2*ratioIndB[i]))
806 print("For Es/N0(db)={}".format(ratioIndB[i]))
807 print("Empirical BER: ", BERempirical)
808 print("Theoretical BER: ", BERtheoretical)

```

```

586 # (v) Constellation Diagrams
587 Xs = np.zeros(int(len(binary)/2)) # Demodulation of the signal
588 Ys = np.zeros(int(len(binary)/2))
589 for time in range(t.size):
590     Xs[int(t[time]/(2*Tb))] += sigWnoise[time]*math.cos(2*pi*fc*t[time])
591     Ys[int(t[time]/(2*Tb))] += sigWnoise[time]*math.sin(2*pi*fc*t[time])
592     ax = figCon.add_subplot(2,1,1+i) # create one subplot
593     ax.set_title("Constellation Diagram of the QPSK Signal (Es/N0 = {}dB)".format(ratioIndB[i]))
594     ax.set_ylabel("Quadrature Component- $\phi_2(t)$ ")
595     ax.set_xlabel("In Phase Component- $\phi_1(t)$ ")
596     ax.grid() # use the grid
597     for bitPair in range(Xs.size):
598         ax.scatter(Xs[bitPair], Ys[bitPair], s = 1, c='b')

```

Υλοποίηση σε Python

(vii) Έχοντας αποδιαμορφώσει τα σήματα, ανακατασκευάζουμε τα αρχεία κειμένου:

Input Text:

The random noise considered is that which arises from shot effect in vacuum tubes or from thermal agitation of electrons in resistors. [...] When a noise voltage or a noise voltage plus a signal is applied to a non-linear device, such as a square-law or linear rectifier, the output will also contain noise. [...] The shot effect in vacuum tubes is a typical example of noise. It is due to fluctuations in the intensity of the stream of electrons flowing from the cathode to the anode.

For Es/N0(db)=5

Output Text:

The random noise considered is that which arises from shot effect in vacuum tubes or from thermal agitation of electrons in resistors. [...] When a noise voltage or a noise voltage plus a signal is applied to a non-linear device, such as a square-law or linear rectifier, the output will also contain noise. [...] The shot effect in vacuum tubes is a typical example of noise. It is due to fluctuations in the intensity of the stream of electrons flowing from the cathode to the anode.

For Es/N0(db)=15

Output Text:

The random noise considered is that which arises from shot effect in vacuum tubes or from thermal agitation of electrons in resistors. [...] When a noise voltage or a noise voltage plus a signal is applied to a non-linear device, such as a square-law or linear rectifier, the output will also contain noise. [...] The shot effect in vacuum tubes is a typical example of noise. It is due to fluctuations in the intensity of the stream of electrons flowing from the cathode to the anode.

Παρατηρούμε ότι τα ληφθέντα κείμενα δεν έχουν κανένα λάθος σε σύγκριση με το αρχικό κείμενο προς διαμόρφωση.

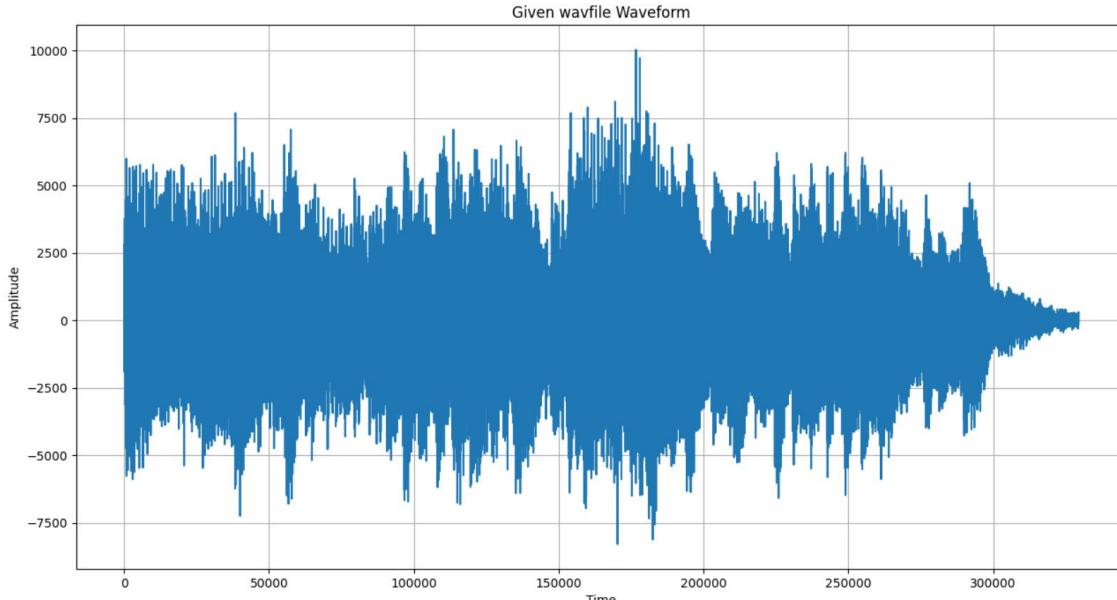
Οι εμπειρικές τιμές BER διαφέρουν από τις θεωρητικές αφού ο αριθμός των bits δεν είναι αρκετά μεγάλος ώστε η θεωρητική πιθανότητα σφάλματος να μπορεί να προσεγγιστεί από την εμπειρική συγχότητα σφάλματος.

Υλοποίηση σε Python

```
731 fOut = open("reconstructedText{}dB.txt".format(ratioIndB[i]), "w")
732 out = ''
733 for bit in range(int(len(binaryRecon)/8)):
734     temp = int(binaryRecon[8*bit:8*bit+8], 2) # convert binary to chars
735     out += temp.to_bytes((temp.bit_length() + 7) // 8, 'big').decode()
736 print("Output Text:")
737 print(out) # print reconstructed text to terminal
738 fOut.write(out) # write reconstructed text to output file
739 fOut.close()
740 fig.tight_layout()
741 figCon.tight_layout()
```

5^ο Ερώτημα:

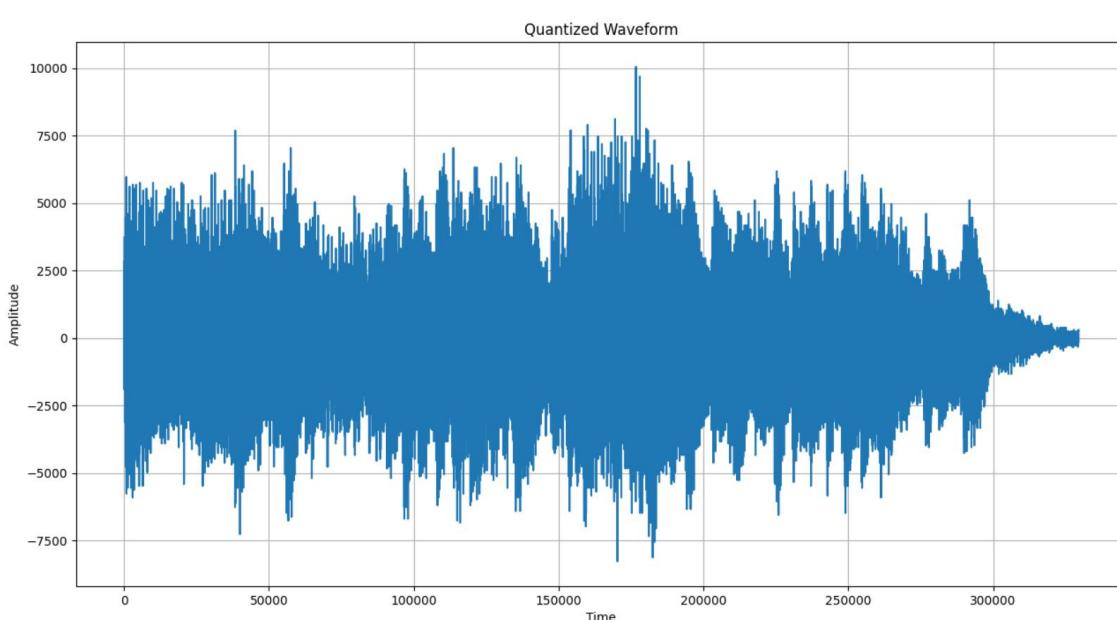
(α') Διαβάζουμε το αρχείο soundfile1_lab2.wav και παρουσιάζουμε σε διάγραμμα την κυματομορφή του σήματος που αναπαριστά:

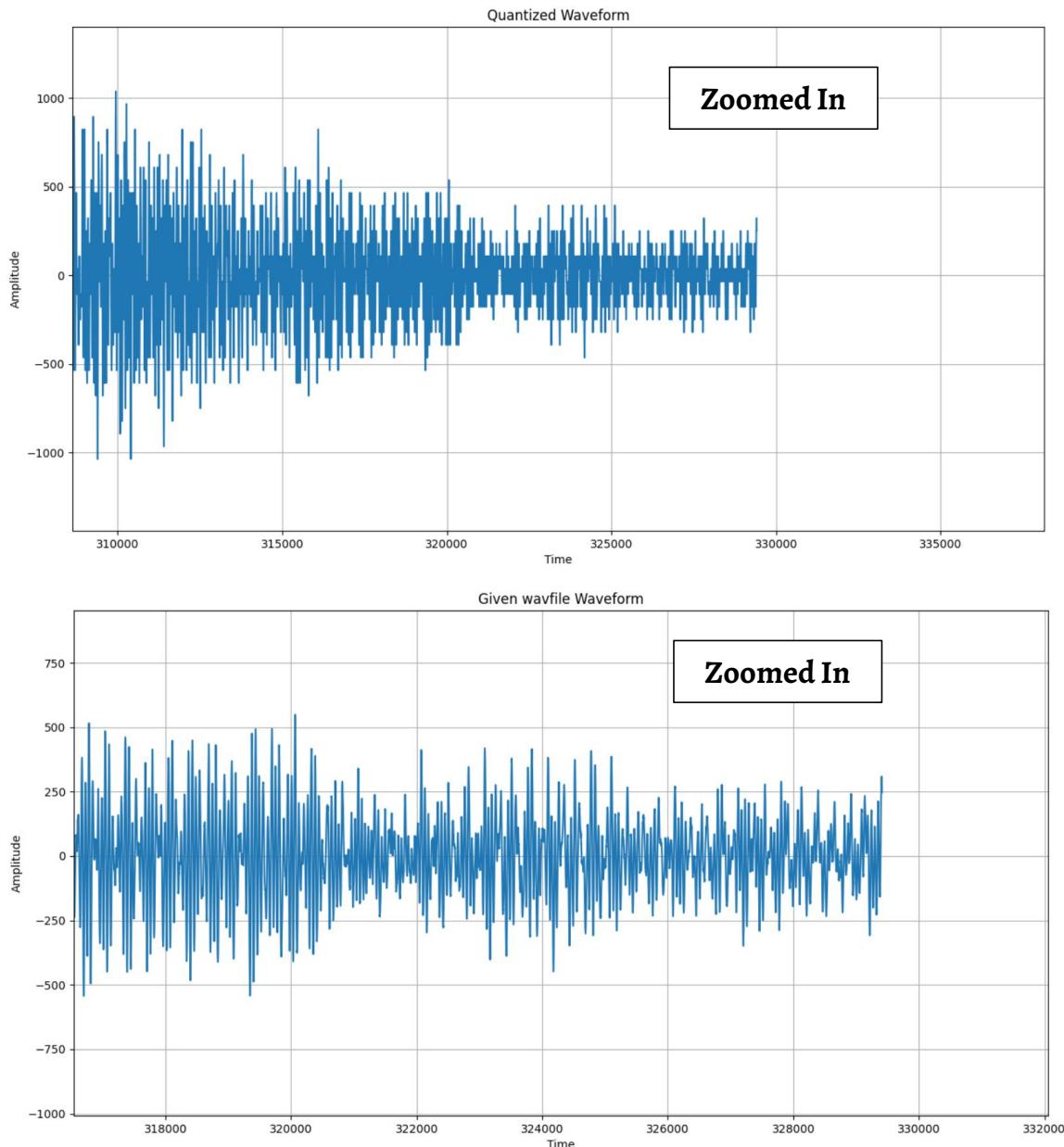


Υλοποίηση σε Python

```
746 from scipy.io.wavfile import read
747
748 input_data = read("soundfile1_lab2.wav")
749 audio = input_data[1]
750 fig = plt.figure(20) # create a new figure
751 ax = fig.add_subplot(111) # create one subplot
752 plt.title("Given wavfile Waveform")
753 plt.grid() # use the grid
754 plt.plot(audio)
755 plt.ylabel("Amplitude")
756 plt.xlabel("Time")
```

(β') Κβαντίζουμε το σήμα χρησιμοποιώντας ομοιόμορφο κβαντιστή 8 ψηφίων:





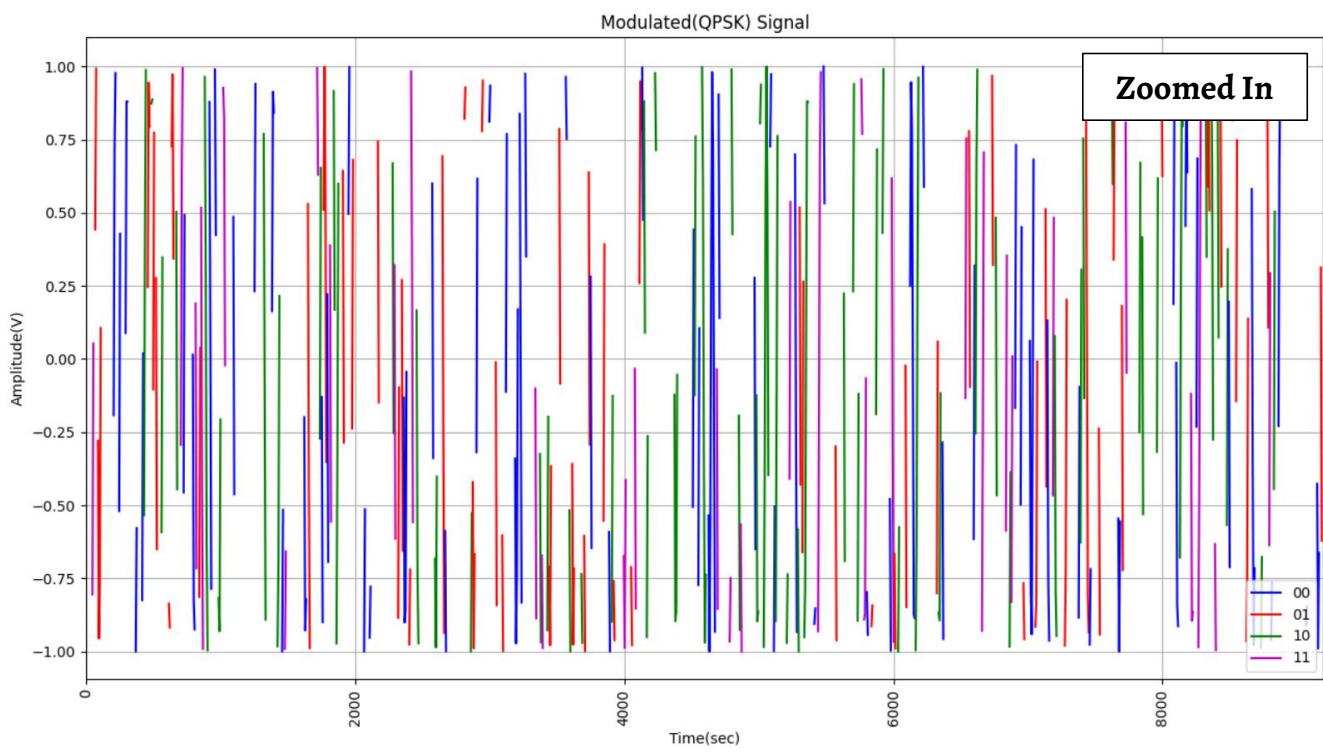
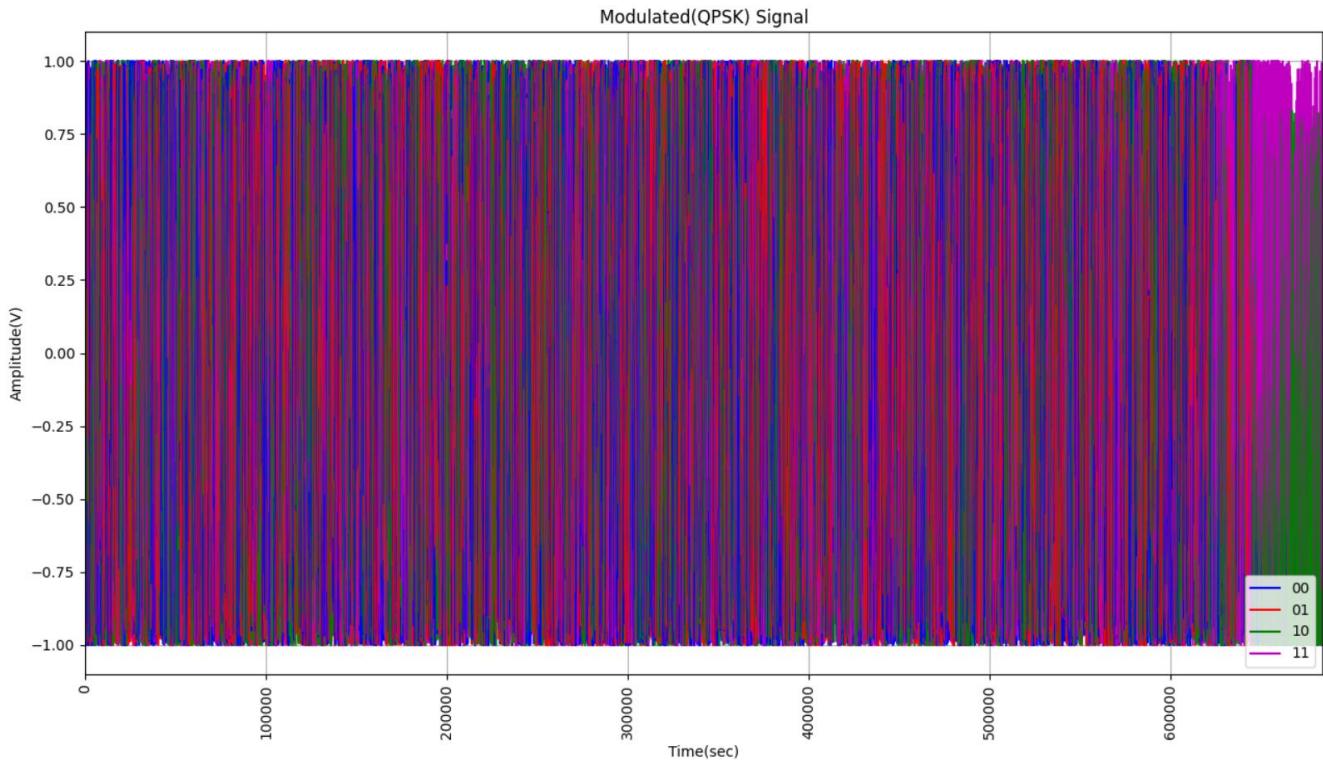
Παρατηρούμε την διαφορά κβαντισμένου σήματος και μη. Εισάγεται στο σήμα σφάλμα κβάντισης.

```

760 bits = 8
761 levels = 2**bits # Levels of quantization
762 maxValue = max(audio)
763 minValue = min(audio)
764 D = (maxValue-minValue)/levels # quantization step = 2*maxValue
765 def quantize(x): # apply classification rule
766     temp = np.floor(x/D)
767     return temp-1 if x == maxValue else temp
768
769 vquantize = np.vectorize(quantize)
770 # make the quantize() function an element-wise one
771
772 q = D*(vquantize(audio) + 1/2) # output values of quantizer
773 fig = plt.figure(21) # create a new figure
774 ax = fig.add_subplot(111) # create one subplot
775 plt.title("Quantized Waveform")
776 plt.grid() # use the grid
777 plt.plot(q)
778 plt.ylabel("Amplitude")
779 plt.xlabel("Time")
780
781 qLvl = vquantize(audio) # quantization_Level
782 qLvl = [qLvl[i]+abs(quantize(minValue)) for i in range(qLvl.size)]
783 # we want positive quantization Levels
784 qBinary = ''
785 for lvl in qLvl:
786     temp = bin(int(lvl))
787     temp = temp[0]+temp[2:]
788     qBinary += temp.zfill(8)
    
```

Υλοποίηση σε Python

(γ') Διαμορφώνουμε το κβαντισμένο σήμα χρησιμοποιώντας διαμόρφωση QPSK θεωρώντας απεικόνιση με κωδικοποίηση Gray και σύμβολα πλάτους IV. Λόγω περιορισμού της υπολογιστικής ισχύος ο χρόνος δεν είναι “τημηματοποιημένος” αρκετά ώστε να προσεγγιστεί η συνεχής φύση του. Έτσι, όπως φαίνεται παρακάτω στην “Zoomed In” εικόνα, το σήμα παραμορφώνεται.

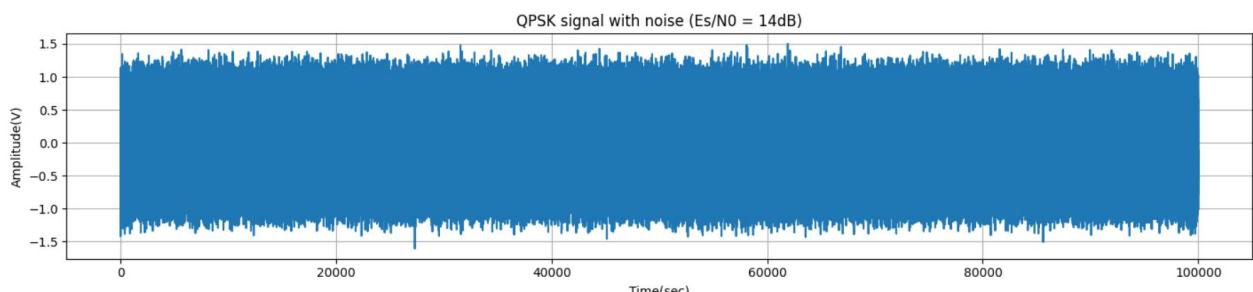
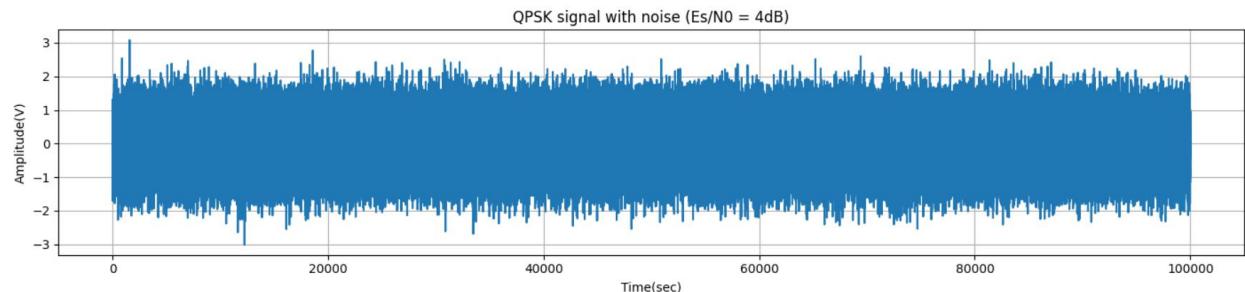


```

791 fc = 1
792 Tb = 0.25
793 A = 1
794 def QPSK_bits(t):
795     return str(qBinary[2*int(t/(2*Tb))])+str(qBinary[2*int(t/(2*Tb))+1])
796
797 def QPSK(t): # which formula to use (gray coded)
798     if QPSK_bits(t) == '00':
799         return A*math.cos(2*pi*fc*t)
800     elif QPSK_bits(t) == '01':
801         return A*math.sin(2*pi*fc*t)
802     elif QPSK_bits(t) == '11':
803         return -A*math.cos(2*pi*fc*t)
804     elif QPSK_bits(t) == '10':
805         return -A*math.sin(2*pi*fc*t)
806
807 vQPSK = np.vectorize(QPSK)
808 # make the QPSK() function an element-wise one
809
810 t = np.linspace(0, len(qBinary)*Tb, 100000, endpoint = False)
811
812 fig = plt.figure(22) # create a new figure
813 ax = fig.add_subplot(111) # create one subplot
814 plt.title("Modulated(QPSK) Signal")
815 prevBits = QPSK_bits(t[0])
816 lastPlotted = 0
817 colors = {'00':'b', '01':'r','10':'g','11':'m'}
818 for i in colors:
819     ax.plot([],[], c=colors[i], label=i)
820 # Plot color-legends
821
822 for i in range(1, t.size):
823     if QPSK_bits(t[i]) != prevBits or i == t.size - 1:
824         ax.plot(t[lastPlotted:i], vQPSK(t[lastPlotted:i]), c=colors[prevBits])
825         prevBits = QPSK_bits(t[i])
826         lastPlotted = i
827 # we plot different colors for each gray code
828
829 plt.ylabel("Amplitude(V)")
830 plt.xlabel("Time(sec)")
831 plt.grid() # use the grid
832 ax.set_xlim(left = 0, right = len(qBinary)*Tb)
833 ax.legend(loc = 'lower right')

```

(δ') Παράγουμε θόρυβο AWGN και τον προσθέτουμε στο σήμα QPSK για $E_s/N_0 = 4$ dB και $E_s/N_0 = 14$ dB:



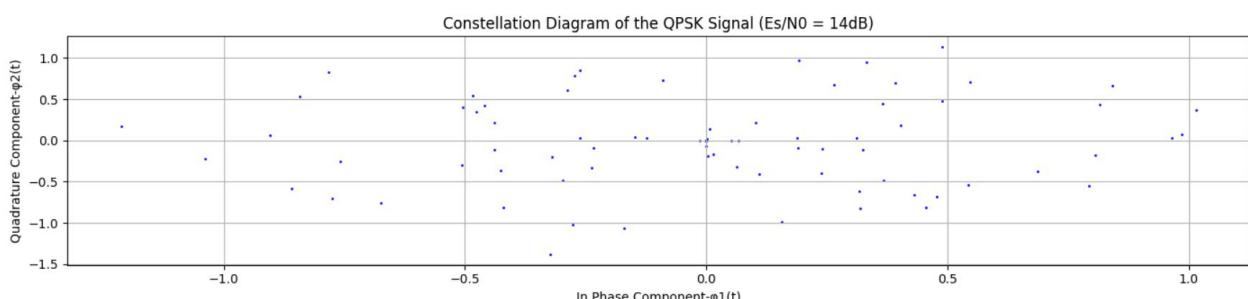
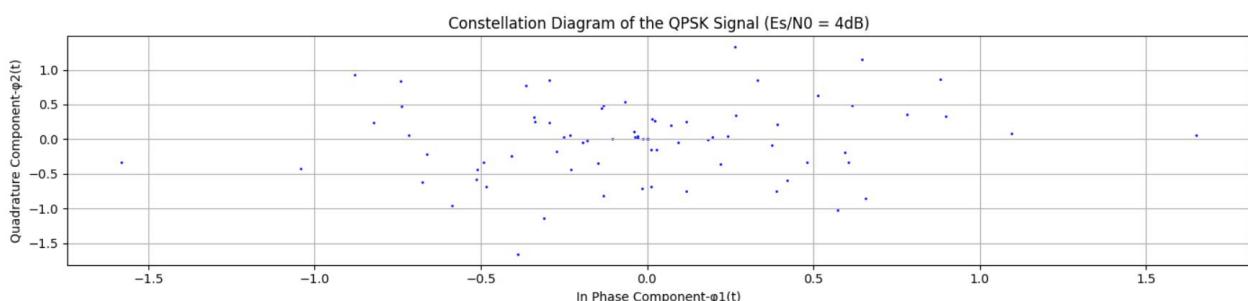
```

838 ratioIndB = np.array([4, 14]) # ratio(dB) = 10Log(Es/(N0/2))
839 ratio = 10**ratioIndB/10
840 SNR = 2*ratio # we have N0/2 one-sided noise power spectral density
841 eQPSK = A*A*Tb # Energy of bit in QPSK modulation
842 noisePower = eQPSK/Tb/SNR # power of given QPSK signal = eQPSK/Tb
843 noiseMean = 0
844 fig = plt.figure(23) # create a new figure
845 figCon = plt.figure(24) # create a new figure
846 for i in range(2):
847     noise = np.random.normal(noiseMean, noisePower[i]**(1/2.0), t.size)
848     ax = fig.add_subplot(2,1,1+i) # create one subplot
849     ax.set_title("QPSK signal with noise (Es/N0 = {}dB)".format(ratioIndB[i]))
850     ax.set_ylabel("Amplitude(V)")
851     ax.set_xlabel("Time(sec)")
852     ax.grid() # use the grid
853     ax.set_xlim(left = 0, right = len(qBinary)*Tb)
854     sigWnoise = noise + vQPSK(t) # noise added to original signal
855     ax.plot(t, sigWnoise)

```

Υλοποίηση σε Python

(ε') Αποδιαμορφώνουμε και παρουσιάζουμε το διάγραμμα αστερισμών για τα σήματα που προέκυψαν:



Υλοποίηση σε Python

```
1009 # (v) Constellation Diagrams
1010 Xs = np.zeros(int(len(qBinary)/2)) # Demodulation of the signal
1011 Ys = np.zeros(int(len(qBinary)/2))
1012 for time in range(t.size):
1013     Xs[int(t[time]/(2*Tb))] += sigWnoise[time]*math.cos(2*pi*fc*t[time])
1014     Ys[int(t[time]/(2*Tb))] += sigWnoise[time]*math.sin(2*pi*fc*t[time])
1015 ax = figCon.add_subplot(2,1,1+i) # create one subplot
1016 ax.set_title("Constellation Diagram of the QPSK Signal (Es/N0 = {}dB)".format(ratioIndB[i]))
1017 ax.set_ylabel("Quadrature Component- $\phi_2(t)$ ")
1018 ax.set_xlabel("In Phase Component- $\phi_1(t)$ ")
1019 ax.grid() # use the grid
1020 for bitPair in range(10000):
1021     ax.scatter(Xs[bitPair], Ys[bitPair], s = 1, c='b')
```

(στ') Υπολογίζουμε την πιθανότητα εσφαλμένου ψηφίου BER εμπειρικά και θεωρητικά:

For Es/N0(db)=4
Empirical BER: 0.46918755635576065
Theoretical BER: 0.012500818040737549

For Es/N0(db)=14
Empirical BER: 0.4683165538698962
Theoretical BER: 6.81010803305071e-13

```
1026 phases = [0, pi/2, pi, 3*pi/2]
1027 codes = ['00', '01', '11', '10']
1028 sI = [math.cos(k) for k in phases]
1029 sQ = [math.sin(k) for k in phases]
1030 binaryRecon = ''
1031 for bitPair in range(Xs.size):
1032     min = 10000 # one big number for min var init
1033     minJ = 0
1034     for j in range(4): # find the min distance among original signals
1035         if (Xs[bitPair]-sI[j])**2 + (Ys[bitPair]-sQ[j])**2 < min:
1036             min = (Xs[bitPair]-sI[j])**2 + (Ys[bitPair]-sQ[j])**2
1037             minJ = j
1038     binaryRecon += codes[minJ] # s_j decided
1039
1040 errors = 0
1041 for bit in range(len(binaryRecon)):
1042     if binaryRecon[bit] != qBinary[bit]:
1043         errors += 1
1044 BERempirical = errors/len(binaryRecon)
1045 BERtheoretical = Q(math.sqrt(2*ratio[i]))
1046 print("For Es/N0(db)={}".format(ratioIndB[i]))
1047 print("Empirical BER: ", BERempirical)
1048 print("Theoretical BER: ", BERtheoretical)
1049
```

Υλοποίηση σε Python

Οι υπολογισμοί BER (θεωρητικοί-εμπειρικοί) διαφέρουν αισθητά. Οι λόγοι αναλύθηκαν και στα προηγούμενα ερωτήματα. Έχουμε σφάλμα κβάντισης αλλά και μη συνεχή χρόνο. Το μεγαλύτερο σφάλμα το εισάγει η διακριτή προσέγγιση που έχουμε κάνει για τον χρόνο καθώς η υπολογιστική μας ισχύς δεν μας επιτρέπει να επιλέξουμε αρκετά μικρό χρονικό βήμα. Έτσι το σήμα QPSK παραμορφώνεται εμφανώς.