

Άσκηση 1

Ερώτημα Α:

Η πιθανότητα σφάλματος bit του συστήματος PAM δίνεται από τη σχέση :

$$P_b = 2 \frac{M-1}{M \log_2 M} Q\left(\sqrt{\frac{6 * SNR_b * \log_2 M}{M^2 - 1}}\right)$$

Γνωρίζω ότι :

$$Q^{-1}(y) = \sqrt{2} * \operatorname{erfc}^{-1}(2 * y) \quad (1)$$

Ορίζω,

$$x = \sqrt{\frac{6 * SNR_b * \log_2 M}{M^2 - 1}}$$

Και γράφω την εξίσωση ως :

$$P_b = 2 \frac{M-1}{M \log_2 M} Q(x)$$

Με βάση την εξ. (1) :

$$x = Q^{-1}\left(P_b * \frac{M \log_2 M}{M-1}\right) = \sqrt{2} * \operatorname{erfc}^{-1}\left(P_b * \frac{M \log_2 M}{M-1}\right)$$

Αντικαθιστώ το x και λύνω ως προς SNR_b :

$$\sqrt{\frac{6 * SNR_b * \log_2 M}{M^2 - 1}} = \sqrt{2} * \operatorname{erfc}^{-1}\left(P_b * \frac{M \log_2 M}{M-1}\right)$$

$$SNR_b = \frac{M^2 - 1}{3 \log_2 M} \left[\operatorname{erfc}^{-1}\left(P_b * \frac{M \log_2 M}{M-1}\right) \right]^2$$

$$SNR_{db} = 10 \log_{10}(SNR_b)$$

Ερώτημα Β:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erfcinv

# Υπολογίζω το SNRdb χρησιμοποιώντας την erfcinv της βιβλιοθήκης
def SNRdb(M,Pb):
    a=np.sqrt(2)*erfcinv(Pb*M*np.log2(M)/(M-1))
    SNR=np.power(a,2)*(np.power(M,2)-1)/6/np.log2(M)
    return 10*np.log10(SNR)

# Το τελευταίο ψηφίο του AM
last_digit=9;

# Πιθανότητα σφάλματος Pb
P=1/np.power(10,(last_digit+2))
print('Pb=',P)

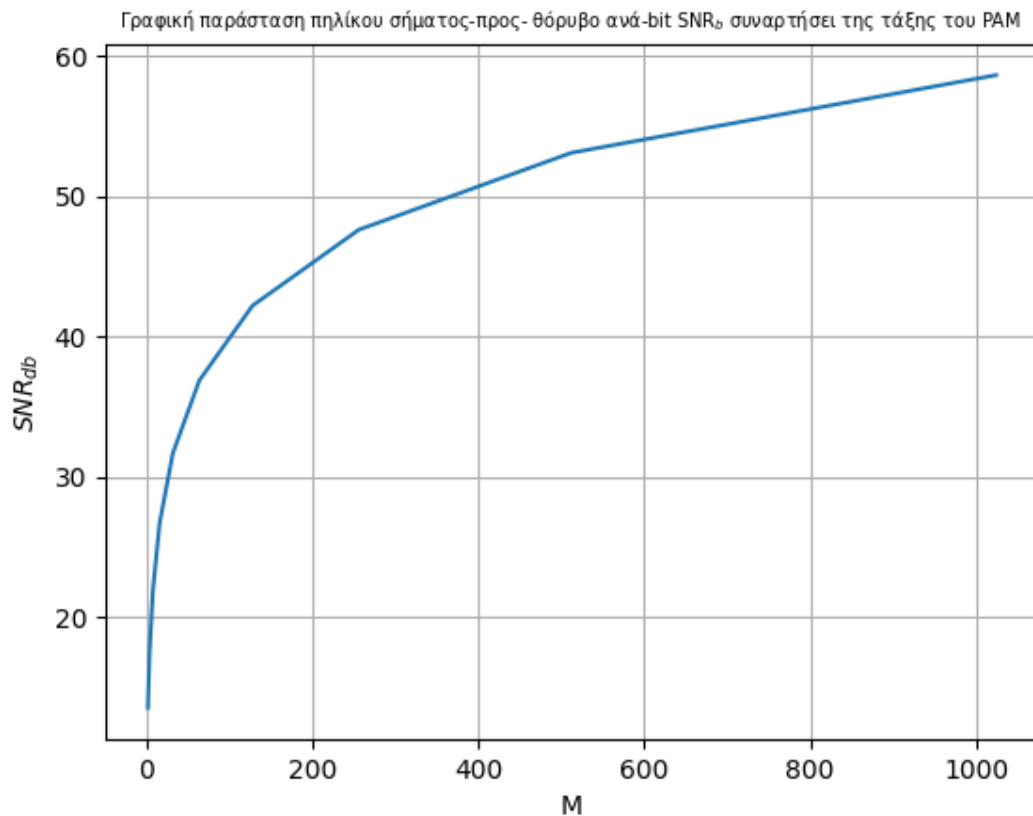
# Υπολογίζω τις δυνάμεις του 2^m και τις αποθηκεύω στη μεταβλητή M
M = [np.power(2,i) for i in range(1, 10+1)]
print('M=',M,'\n');

# Καλώ τη συνάρτηση για τον υπολογισμό του SNRdb για τις διάφορες τιμές M και P, και τις
# αποθηκεύω στη λίστα SNr
SNr=[SNRdb(x,P) for x in M ]

# Γραφικές των αποτελεσμάτων
plt.plot(M, SNr,linestyle = 'solid')
plt.title("SNR$_{db}$=f(M)")
plt.ylabel("$SNR_{db}$")
plt.xlabel("M")
plt.grid()

plt.savefig('snr.png', dpi='figure')
plt.show()
```

Για $M=2$ έως 1024 και τελευταίο ψηφίο $AM=9 \rightarrow Pb=10^{-11}$.
Το αποτέλεσμα του παραπάνω κώδικα φαίνεται στο παρακάτω διάγραμμα.



Ερώτημα Γ:

Στο παραπάνω διάγραμμα φαίνεται πως όσο αυξάνεται η τάξη M του PAM, αυξάνεται η φασματική απόδοση αλλά το σύστημα γίνεται περισσότερο ευαίσθητο στον θόρυβο. Υψηλότερες τιμές " M " επιτρέπουν τη μετάδοση περισσότερων bits ανά σύμβολο, αλλά καθιστούν το σύστημα περισσότερο ενεργοβόρο καθώς το SNR_b μεγαλώνει, απαιτώντας υψηλότερη ενέργεια στον δέκτη.

Άσκηση 2

```
import commlib as cl2
import ppm_wave as ppm
import numpy as np
import matplotlib.pyplot as plt

"""Ορισμός αρχικών δεδομένων / μεταβλητών
"""

M=2
TS=1/1e+11
name='Maria Kollia'

""" Βήμα 1ο
    Η συνάρτηση δέχεται το ονοματεπώνυμο στα αγγλικά, χωρίζει το string σε χαρακτήρες.
    Στη συνέχεια, κάθε χαρακτήρας μετατρέπεται σε μία σειρά από 8bits.
"""

def str_to_bits_array(str):
    binary = ".join(format(ord(i), '08b') for i in str)
    return np.array( [ int(x) for x in binary ] ).astype(int)

"""
Βήμα 2ο:
    Σε περίπτωση όπου στην τελευταία ομάδα bits που μεταδίδουμε απομένουν λιγότερα
    από bits συμπληρώνουμε μηδενικά bits ώστε να έχουμε bits.
"""

def pad_zeroes(bits_array,M):
    if len(bits_array)%np.log2(M)>0:
        add_zeros=np.log2(M)-len(bits_array)%np.log2(M)
        bits_array=np.pad(bits_array,(0, int(add_zeros)), mode='constant', constant_values=0)

    return bits_array

binary_name=str_to_bits_array(name)
bits=pad_zeroes(binary_name,M)

# Μετατροπή των bits σε σύμβολα gray για τον υπολογισμό της κυματομορφής M-PPM και
ομαδοποίηση κατά log2(M) bits.
```

```

fmap=cl2.pam_gray_forward_map(M)
symbols, bitgroups = cl2.bits_to_symbols(bits, fmap, return_bits = True)
print('Bit groups:')
print(bitgroups)
print('Encoded symbols:')
print(symbols)

#Υπολογισμός κυματομορφής μέσω της συνάρτησης ppm_waveform από το αρχείο ppm.py
t, x = ppm.ppm_waveform(symbols, TS,M)

##### Αποθήκευση Διαγράμματος #####
cl2.plot_pam(t, x, M, bits, TS, dy = 1, plot_type = '-')
plt.title("M=%i" %M,loc='right')
plt.rc('font', size=6)
figure = plt.gcf() # get current figure
figure.set_size_inches(14, 10)
plt.savefig("M"+str(M)+".png", dpi = 200)

```

Η συνάρτηση ppm_waveform

```

import numpy as np
"""
    Χρησιμοποιείται ο παλμός default_pulse ώστε να υπολογιστεί ο παλμός κάθε ομάδα bit σε
    μία περίοδο.
    p(t)=1, T1<t<T2, else p(t)=0
    """
def default_pulse(t, T1,T2):
    i = np.where( (t >= T1) & (t<T2) )
    x = np.zeros(t.size)
    x[i] = 1
    return x

def ppm_waveform(k,T,M,p_callable=default_pulse,samples=200):
    # Φτιάχνω ένα διάνυσμα το οποίο περιέχει (αριθμό bits)*samples αριθμό χρονικών
    στιγμών για κάθε ομάδα bit.
    # από 0 έως (αριθμό bits)* T, ώστε να επαρκεί ο χρόνος που χρειάζεται για να χωρέσει
    # ο κάθε παλμός μέσα σε μία περίοδο.
    t=np.linspace(0,len(k)*T,len(k)*len(k)*samples+1)
    a=1
    wave=np.zeros(len(t))
    # Εφόσον γνωρίζω πόσα χρονικά στοιχεία αντιστοιχούν σε μία περίοδο TS, χρησιμοποιώ
    το μέρος του t που αντιστοιχεί στην ομάδα bit που θέλω να μεταδώσω.
    idx=np.arange((a-1)*len(k)*samples,a*len(k)*samples,1)
    for i in range(0,len(k)):
        # search in this region
        wave[idx]=(p_callable(t[idx],k[a-1]*T/M+(a-1)*T, (k[a-1]+1)*T/M+(a-1)*T ))

```

```
# κάθε φορά που αυξάνει το α, μετακινούμε κατά μία περίοδο TS ώστε να μεταδώσω
τον επόμενο παλμό.
a=a+1
idx=np.arange((a-1)*len(k)*samples,a*len(k)*samples,1)

return t, wave
```

Παρακάτω φαίνονται οι γραφικές παραστάσεις για $M=2$, $M=16$.

Βλέπουμε πως όσο αυξάνουμε τον αριθμό M μπορούμε να μεταδώσουμε περισσότερα bits πληροφορίας με κάθε παλμό, αφού για $M=16$ έχουμε 16 διαφορετικές ομαδοποιήσεις σε αντίθεση με $M=2$ που έχουμε μόνο 2.

Όμως, όσο αυξάνεται το M τόσο το σύστημα γίνεται ευαίσθητο σε θόρυβο.

Επομένως, εάν θέλουμε να μεταδώσουμε μία σειρά από bits με γρήγορο ρυθμό θα διαλέξουμε $M=16$, εάν θέλουμε το σύστημα να επηρεάζεται λιγότερο από θόρυβο τότε $M=2$.

