ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

**Διαδραστικές ασκήσεις Ψηφιακών Επικοινωνιών στη γλώσσα Python**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΦΡΑΓΚΟΥΛΟΠΟΥΛΟΥ ΛΑΜΠΡΟΥ

Επιβλέπων

Μήτρου Νικόλαος

Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

**Διαδραστικές ασκήσεις Ψηφιακών Επικοινωνιών στη γλώσσα Python**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΦΡΑΓΚΟΥΛΟΠΟΥΛΟΥ ΛΑΜΠΡΟΥ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την\_\_\_\_\_\_\_\_\_  
  
.............................. .............................. ..............................

Ρουσσάκη Ιωάννα  
Καθηγητής Ε.Μ.Π.

Σύκας Ευστάθιος  
Καθηγητής Ε.Μ.Π.

Μήτρου Νικόλαος  
Καθηγητής Ε.Μ.Π.

Μήτρου Νικόλαος  
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2024

.............................   
Φραγκουλόπουλος Ν. Λάμπρος   
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Φραγκουλόπουλος Λάμπρος, 2024   
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

# Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ψηφιακή επεξεργασία σήματος με εφαρμογή στις τηλεπικοινωνίες, καλύπτοντας σημαντικές θεωρητικές και πρακτικές πτυχές. Ξεκινώντας με μια εισαγωγή στον ρόλο της ψηφιακής επεξεργασίας σήματος στις τηλεπικοινωνίες, η εργασία διερευνά την εξομοίωση αναλογικών διαμορφώσεων και τις βασικές αρχές της βέλτιστης ψηφιακής αναγνώρισης μέσω προσαρμοσμένων φίλτρων.

Στη συνέχεια, αναλύονται τα φασματικά χαρακτηριστικά των ψηφιακών κυματομορφών και η μορφοποίησή τους με φίλτρα Nyquist, τα οποία διασφαλίζουν την ελαχιστοποίηση των παρεμβολών μεταξύ συμβόλων. Ιδιαίτερη έμφαση δίνεται στη μελέτη των διαμορφώσεων L-ASK, QAM και PSK, οι οποίες είναι ευρέως διαδεδομένες σε σύγχρονα συστήματα επικοινωνιών, όπως και στις διαμορφώσεις FSK και MSK.

Στο πλαίσιο του μαθήματων των Ψηφιακών Επικοινωνιών Ι, οι εργαστηριακές ασκήσεις που αρχικά είχαν γραφτεί σε MATLAB μετατράπηκαν σε Python, προσφέροντας μεγαλύτερη ευελιξία και διαδραστικότητα. Η χρήση του Jupyter Book για τη δημιουργία μιας διαδραστικής ιστοσελίδας ενισχύει την εμπειρία μάθησης για το μάθημα των ψηφιακών επικοινωνιών, επιτρέποντας στους φοιτητές να εξερευνήσουν τις έννοιες μέσα από πρακτικές εφαρμογές και διαδραστικά παραδείγματα.

Η εργασία αυτή αποτελεί μια ολοκληρωμένη προσέγγιση στη μελέτη της ψηφιακής επεξεργασίας σήματος και των εφαρμογών της στις τηλεπικοινωνίες, συνδυάζοντας θεωρητική ανάλυση με σύγχρονα εργαλεία προγραμματισμού και εκπαιδευτικές πλατφόρμες.

**Λέξεις-Κλειδιά:** Ψηφιακή Επεξεργασία Σήματος, Ανάλυση Φάσματος, Τηλεπικοινωνίες, Ψηφιακή Διαμόρφωση, L-ASK, QAM, PSK, FSK, MSK, Προσαρμοσμένα Φίλτρα, Φίλτρα Nyquist, Απόδοση Διαύλου, OFDM, , Jupyter Book, Python

**.**

# Abstract

This thesis focuses on digital signal processing with applications in telecommunications, covering significant theoretical and practical aspects. Beginning with an introduction to the role of digital signal processing in telecommunications, the thesis explores the simulation of analog modulations and the fundamental principles of optimal digital recognition through matched filters.

Next, it analyzes the spectral characteristics of digital waveforms and their shaping using Nyquist filters, which ensure the minimization of intersymbol interference. Special emphasis is placed on the study of L-ASK, QAM, and PSK modulations, which are widely used in modern communication systems, as well as FSK and MSK modulations.

As part of the Digital Communications I course, the lab exercises that were initially written in MATLAB were converted into Python, offering greater flexibility and interactivity. The use of Jupyter Book to create an interactive webpage enhances the learning experience for the Digital Communications course, allowing students to explore concepts through practical applications and interactive examples.

This thesis represents a comprehensive approach to the study of digital signal processing and its applications in telecommunications, combining theoretical analysis with modern programming tools and educational platforms.

**Keywords:** Digital Signal Processing, Spectrum Analysis, Telecommunications, Digital Modulation, L-ASK, QAM, PSK, FSK, MSK, Matched Filters, Nyquist Filters, Channel Performance, OFDM, Jupyter Book, Python

# Ευχαριστίες

Αρχικά, θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες προς τον επιβλέποντα καθηγητή, κ. Νικόλαο Μήτρου, για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου τη διπλωματική αυτή εργασία. Μέσω αυτής, μου δόθηκε η ευκαιρία να εμβαθύνω σε ένα εξαιρετικά ενδιαφέρον και χρήσιμο αντικείμενο, όπως είναι η διαδραστική ανάπτυξη των ασκήσεων της Ψηφιακής Επικοινωνίας σε γλώσσα Python. Επίσης, θα ήθελα να ευχαριστήσω τον ίδιο αλλά και την κ. Κωνσταντία Σακκά για τη συνεχή υποστήριξη και καθοδήγηση που μου παρείχε καθ' όλη τη διάρκεια της εκπόνησης της εργασίας.

Η διπλωματική αυτή εργασία αποτελεί το επιστέγασμα των προπτυχιακών μου σπουδών στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Στο τέλος αυτής της πορείας, αισθάνομαι την ανάγκη να ευχαριστήσω θερμά τη μητέρα μου, Δήμητρα, η οποία απεβίωσε κατά το πρώτο έτος των σπουδών μου, καθώς και τον πατέρα μου, Νικόλαο, και τον αδελφό μου, Ιωάννη, για την αδιάκοπη στήριξή τους σε όλα τα χρόνια της ακαδημαϊκής μου πορείας.

Θα ήθελα επίσης να εκφράσω την ευγνωμοσύνη μου προς όλους τους φίλους που γνώρισα κατά τη διάρκεια των φοιτητικών μου χρόνων. Μαζί τους μοιραστήκαμε δύσκολες στιγμές, απαιτητικές εργασίες και εξεταστικές περιόδους, και δημιουργήσαμε αξέχαστες αναμνήσεις που θα με συνοδεύουν σε όλη μου τη ζωή. Σας ευχαριστώ όλους από καρδιάς.

# Οργάνωση του τόμου

Η παρούσα διπλωματική εργασία είναι οργανωμένη σε επτά κύρια κεφάλαια, τα οποία καλύπτουν θεωρητικές και πρακτικές πτυχές της ψηφιακής επεξεργασίας σήματος και των εφαρμογών της στις τηλεπικοινωνίες.

Στο πρώτο κεφάλαιο παρουσιάζεται η εισαγωγή της εργασίας, όπου καθορίζονται οι στόχοι και η συνολική της δομή. Παράλληλα, αναλύεται η σημασία της διαδραστικής εκπαίδευσης και η χρήση της γλώσσας Python για την υλοποίηση προσομοιώσεων. Το δεύτερο κεφάλαιο περιλαμβάνει την ανασκόπηση της βιβλιογραφίας, με έμφαση στις βασικές αρχές της ψηφιακής επεξεργασίας σήματος, όπως η δειγματοληψία, οι τεχνικές διαμόρφωσης σημάτων (QAM, PSK, FSK, MSK), και η χρήση προσαρμοσμένων και φίλτρων Nyquist για τη βελτιστοποίηση των συστημάτων επικοινωνίας. Στο τρίτο κεφάλαιο περιγράφεται η μεθοδολογία που ακολουθήθηκε για την ανάπτυξη των διαδραστικών ασκήσεων. Ιδιαίτερη έμφαση δίνεται στη μετάβαση από το MATLAB στην Python και στη χρήση του Jupyter Book για τη δημιουργία μιας διαδραστικής εκπαιδευτικής πλατφόρμας. Το τέταρτο κεφάλαιο επικεντρώνεται στην ανάλυση των αποτελεσμάτων των εργαστηριακών ασκήσεων, παρουσιάζοντας τις επιδόσεις των συστημάτων ψηφιακής επικοινωνίας που προσομοιώθηκαν με βάση διαφορετικές παραμέτρους, όπως ο σηματοθορυβικός λόγος (SNR) και η πιθανότητα λάθους (BER). Στο πέμπτο κεφάλαιο εξετάζονται πιθανές μελλοντικές επεκτάσεις της εργασίας, όπως η βελτιστοποίηση των αλγορίθμων που χρησιμοποιήθηκαν και η επέκταση της πλατφόρμας με επιπλέον διαδραστικές δυνατότητες. Το έκτο κεφάλαιο συνοψίζει τα κύρια αποτελέσματα της εργασίας, επισημαίνοντας τις καινοτομίες που προκύπτουν από τη χρήση της Python και του Jupyter Book για την εκπαίδευση στις ψηφιακές επικοινωνίες. Το έβδομο κεφάλαιο περιλαμβάνει τη βιβλιογραφία και τις πηγές που χρησιμοποιήθηκαν για την εκπόνηση της εργασίας.

Στο Παράρτημα της διπλωματικής εργασίας παρατίθενται αναλυτικές πληροφορίες και τεχνικές λεπτομέρειες για την υλοποίηση των διαδραστικών ασκήσεων και των εργαλείων που χρησιμοποιήθηκαν. Περιλαμβάνεται ένας πλήρης οδηγός εγκατάστασης του λογισμικού Python και των απαραίτητων βιβλιοθηκών, όπως οι NumPy, SciPy, και Matplotlib, καθώς και οδηγίες για τη χρήση του Jupyter Book. Επίσης, παρέχονται αναλυτικές τεκμηριώσεις των κωδίκων που αναπτύχθηκαν στις εργαστηριακές ασκήσεις, μαζί με οδηγίες για την εκτέλεσή τους, προκειμένου οι φοιτητές να μπορούν να επαναλάβουν τα πειράματα και να αξιοποιήσουν την πλατφόρμα για περαιτέρω μελέτη και πειραματισμό.

**Πίνακας Περιεχομένων**

[Περίληψη 2](#_Toc181363605)

[Abstract 3](#_Toc181363606)

[Ευχαριστίες 4](#_Toc181363607)

[Οργάνωση του τόμου 1](#_Toc181363608)

[Κατάλογος Σχημάτων 10](#_Toc181363609)

[Εισαγωγή 11](#_Toc181363610)

[1.1 Στόχοι της Εργασίας 12](#_Toc181363611)

[1.2 Δομή της Εργασίας 13](#_Toc181363612)

[Ανασκόπηση της Βιβλιογραφίας 14](#_Toc181363613)4

[2.1 Περιγραφή του μαθήματος 15](#_Toc181363614)

[2.1.1 Lab Exercise 1: Εισαγωγή στα Συστήματα Ψηφιακής Μετάδοσης 15](#_Toc181363615)

[2.1.2 Lab Exercise 2: Ψηφιακά Φίλτρα 19](#_Toc181363616)

[2.1.3 Lab Exercise 3: Προσαρμοσμένα Φίλτρα και Διαμόρφωση L-ASK 22](#_Toc181363617)

[2.1.4 Lab Exercise 4: Σηματοδότηση Nyquist και Διαμόρφωση L-ASK 25](#_Toc181363618)

[2.1.5 Lab Exercise 5: Διαμόρφωση QAM και PSK 27](#_Toc181363619)

[2.1.6 Lab Exercise 6: Διαμόρφωση FSK και MSK 30](#_Toc181363620)

[2.2 Εργαλεία που χρησιμοποιήθηκαν 32](#_Toc181363621)

[Μεθοδολογία, Σχεδίαση και Υλοποίηση διαδραστικών στοιχείων 40](#_Toc181363622)

[3.1 Εισαγωγή 41](#_Toc181363623)

[3.2 Μέθοδος υλοποίησης του συνολικού project 41](#_Toc181363624)

[3.3 Τα πρώτα βήματα για το στήσιμο της σελίδας 43](#_Toc181363625)

[3.4 Διαδραστικά στοιχεία που χρησιμοποιήθηκαν 44](#_Toc181363626)

[3.4.1 Καρτέλες (Tabs) 44](#_Toc181363627)

[3.4.2 Αναπτυσσόμενα μενού (Dropdown) 46](#_Toc181363628)

[3.4.3 Πεδίο κειμένου (Text box) 49](#_Toc181363629)

[3.4.4 Ολισθητής (Slider) 53](#_Toc181363630)

[3.4.5 Πλαίσιο επιλογής (Checkbox) 56](#_Toc181363631)

[3.4.6 Loading Animation - Χρονομετρητής (Timer) 59](#_Toc181363632)

[3.4.7 MyST Markdown 63](#_Toc181363633)

[3.5 Bit Error Rate Tool 65](#_Toc181363634)

[Αποτελέσματα 68](#_Toc181363635)

[4.1 Lab Exercise 1: Εισαγωγή στα Συστήματα Ψηφιακής Μετάδοσης 69](#_Toc181363636)

[4.1.1 Αποτελέσματα Προσομοίωσης 69](#_Toc181363637)

[4.1.2 Ανάλυση Επιδόσεων 70](#_Toc181363638)

[4.1.3 Αδυναμίες και Προκλήσεις 71](#_Toc181363639)

[4.2 Lab Exercise 2: Ψηφιακά Φίλτρα 72](#_Toc181363640)

[4.2.1 Αποτελέσματα Προσομοίωσης 72](#_Toc181363641)

[4.2.2 Ανάλυση Επιδόσεων 73](#_Toc181363642)

[4.2.3 Αδυναμίες και Προκλήσεις 75](#_Toc181363643)

[4.3 Lab Exercise 3: Συγχρονισμένα Φίλτρα και Διαμόρφωση L-ASK 76](#_Toc181363644)

[4.3.1 Αποτελέσματα Προσομοίωσης 76](#_Toc181363645)

[4.3.2 Ανάλυση Επιδόσεων 76](#_Toc181363646)

[4.3.3 Αδυναμίες και Προκλήσεις 79](#_Toc181363647)

[4.4 Lab Exercise 4: Σηματοδότηση Nyquist και Διαμόρφωση L-ASK 81](#_Toc181363648)

[4.4.1 Αποτελέσματα Προσομοίωσης 81](#_Toc181363649)

[4.4.2 Ανάλυση Επιδόσεων 82](#_Toc181363650)

[4.4.3 Αδυναμίες και Προκλήσεις 83](#_Toc181363651)

[4.5 Lab Exercise 5: Διαμόρφωση QAM και PSK 84](#_Toc181363652)

[4.5.1 Αποτελέσματα Προσομοίωσης 84](#_Toc181363653)

[4.5.2 Ανάλυση Επιδόσεων 84](#_Toc181363654)

[4.5.3 Αδυναμίες και Προκλήσεις 86](#_Toc181363655)

[4.6 Lab Exercise 6: Διαμόρφωση FSK και MSK 87](#_Toc181363656)

[4.6.1 Αποτελέσματα Προσομοίωσης 87](#_Toc181363657)

[4.6.2 Ανάλυση Επιδόσεων 88](#_Toc181363658)

[4.6.3 Αδυναμίες και Προκλήσεις 89](#_Toc181363659)

[Μελλοντικές επεκτάσεις 90](#_Toc181363660)

[Σύνοψη 92](#_Toc181363661)

[6.1 Συνοπτική παρουσίαση του σκοπού 93](#_Toc181363662)

[6.2 Περίληψη κύριων βημάτων και μεθοδολογίας 93](#_Toc181363663)

[6.3 Βασικά Αποτελέσματα 94](#_Toc181363664)

[6.4 Συμπεράσματα 95](#_Toc181363665)

[Πηγές 96](#_Toc181363666)

[Παράρτημα Ι 98](#_Toc181363667)

[Πλήρης Οδηγός Εγκατάστασης 99](#_Toc181363668)

[Παράρτημα ΙΙ 103](#_Toc181363669)

[Οδηγός χρήσης του Jupyter Book με το Thebe 104](#_Toc181363670)

[**Εκκίνηση Διαδραστικού Κώδικα** 104](#_Toc181363671)

[**Επανεκκίνηση και Εκτέλεση Όλων** 104](#_Toc181363672)

[Documentation: Basic Elements of a Jupyter Book 105](#_Toc181363673)

[Δομή του Jupyter Book 105](#_Toc181363674)

[Αρχεία Περιεχομένου 105](#_Toc181363675)

[Ρυθμίσεις του Jupyter Book 106](#_Toc181363676)

[Διαδραστικότητα με το Thebe 111](#_Toc181363677)

[Documentation of Lab Exercises 113](#_Toc181363678)

[Lab Exercise 1: Εισαγωγή στα Συστήματα Ψηφιακής Μετάδοσης 113](#_Toc181363679)

[Lab Exercise 2: Ψηφιακά Φίλτρα 126](#_Toc181363680)

[Lab Exercise 3: Προσαρμοσμένα Φίλτρα και Διαμόρφωση L-ASK 152](#_Toc181363681)

[Lab Exercise 4: Σηματοδότηση Nyquist και Διαμόρφωση L-ASK 181](#_Toc181363682)

[Lab Exercise 5: Διαμόρφωση QAM και PSK 200](#_Toc181363683)

[Lab Exercise 6: Διαμόρφωση FSK και MSK 225](#_Toc181363684)

# Κατάλογος Σχημάτων

[Σχήμα 1: Δειγματοληψία - Ψηφιοποίηση (Μετασχηματισμός Fourier) 16](#_Toc180582727)

[Σχήμα 3:Δειγματοληψία - Ψηφιοποίηση (Κρουστικές) 16](#_Toc180582728)

[Σχήμα 4:Δειγματοληψία - Ψηφιοποίηση (Πεπερασμένη σειρά xn) 17](#_Toc180582729)

[Σχήμα 6: Αμφίπλευρη και μονόπλευρη αναπαράσταση του μετασχηματισμού Fourier του ημιρονοειδούς σήματος 19](#_Toc180582730)

[Σχήμα 5: Αμφίπλευρη και μονόπλευρη αναπαράσταση ενός ημιτονοειδούς σήματος που έχει πολλαπλασιαστεί με παράθυρο Blackman 21](#_Toc180582731)

[Σχήμα 7: BER συναρτήση του σγματοθορυβικού λόγου 23](#_Toc180582732)

[Σχήμα 10: Φάσμα φίλτρου με εύρος ζώνης W γύρω από τη συχνότητα φέροντος fc 29](#_Toc180582733)

[Σχήμα 12: 1ο Παράδειγμα Καρτελών (Tabs) 44](#_Toc180582734)

[Σχήμα 13: 2ο Παράδειγμα Καρτελών (Tabs)' 45](#_Toc180582735)

[Σχήμα 14: 1ο Παράδειγμα Αναπτυσσόμενου μενού (dropdown) 46](#_Toc180582736)

[Σχήμα 15: 2ο Παράδειγμα Αναπτυσσόμενου μενού (dropdown) 47](#_Toc180582737)

[Σχήμα 16: Παράδειγμα συνδυασμού καρτελών και αναπτυσσόμενου μενού 48](#_Toc180582738)

[Σχήμα 17: 1ο Παράδειγμα πεδίου κειμένου 49](#_Toc180582739)

[Σχήμα 18: 2ο Παράδειγμα πεδίου κειμένου 50](#_Toc180582740)

[Σχήμα 19: 3ο Παράδειγμα πεδίου κειμένου 51](#_Toc180582741)

[Σχήμα 20: Παράδειγμα ελέγχου πεδίου κειμένου 52](#_Toc180582742)

[Σχήμα 21: 1ο Παράδειγμα ολισθητή 53](#_Toc180582743)

[Σχήμα 22: 2ο Παράδειγμα ολισθητή 54](#_Toc180582744)

[Σχήμα 23: 1ο Παράδειγμα πλαισίου επιλογής 56](#_Toc180582745)

[Σχήμα 24: 2ο Παράδειγμα πλαισίου επιλογής 57](#_Toc180582746)

[Σχήμα 25: 1ο Παράδειγμα χρονομετρητή 59](#_Toc180582747)

[Σχήμα 26: 2ο Παράδειγμα χρονομετρητή 60](#_Toc180582748)

[Σχήμα 27: Information box markdown 63](#_Toc180582749)

[Σχήμα 28: Tip box markdown 64](#_Toc180582750)

[Σχήμα 29: Bit Error Rate Tool 65](#_Toc180582751)

[Σχήμα 30: Αναπαράσταση σήματος με είσοδο 3 συχνοτήτων 69](#_Toc180582752)

[Σχήμα 31: Αναπαράσταση φάσματος σήματος 70](#_Toc180582753)

[Σχήμα 32: Αναπαράσταση σήματος και φάσματος αυτού με προσθήκη θορύβου 70](#_Toc180582754)

[Σχήμα 33: Ψηφιακά Φίλτρα 72](#_Toc180582755)

[Σχήμα 34: Απόκριση συχνότητας Equiripple φίλτρου 73](#_Toc180582756)

[Σχήμα 35: Οπτικοποίηση Φίλτρου 74](#_Toc180582757)

[Σχήμα 36: Παράδειγμα ζωνοπερατού φίλτρου 74](#_Toc180582758)

[Σχήμα 37: Παράδειγμα φίλτρου με 2 ζώνες διέλευσης 75](#_Toc180582759)

[Σχήμα 38: Ιστόγραμμα πίνακα x στοιχείων 76](#_Toc180582760)

[Σχήμα 39: Ιστόγραμμα σήματος με θόρυβο 77](#_Toc180582761)

[Σχήμα 40: BER L - ASK διαμόρφωσης 78](#_Toc180582762)

[Σχήμα 41: BER L - ASK διαμόρφωσης με/χωρίς αναστροφή πίνακα 79](#_Toc180582763)

[Σχήμα 42: Αναπάρασταση ορθογωνικού και ημιτονοειδούς παλμού με nsamp σημεία 80](#_Toc180582764)

[Σχήμα 43: Δημιουργία σήματος με Nyquist φίλτρα (Χρονική και φασματική αναπαράσταση) 81](#_Toc180582765)

[Σχήμα 44: BER L - ASK διαμόρφωση 82](#_Toc180582766)

[Σχήμα 45:BER L – ASK (με Gray vs Natural Coding) 83](#_Toc180582767)

[Σχήμα 46Q Αστερισμός MxM QAM 84](#_Toc180582768)

[Σχήμα 47: BER QAM 85](#_Toc180582769)

[Σχήμα 48: BER M - PSK 86](#_Toc180582770)

[Σχήμα 49: Αριθμός λαθών 87](#_Toc180582771)

[Σχήμα 50: BER M - FSK 88](#_Toc180582772)

[Σχήμα 51: BER MSK 89](#_Toc180582773)

[Σχήμα 52: Φασματική πυκνότητα ισχύος 89](#_Toc180582774)

[Σχήμα 53: Κουμπί Interactive code 104](#_Toc180582775)

[Σχήμα 54: Αναμονή για έτοιμο kernel 104](#_Toc180582776)

[Σχήμα 55:Σελίδες του jyputer book 105](#_Toc180582777)

[Σχήμα 56: Αναμονή Διαγράμματος 105](#_Toc180582778)

[Σχήμα 57: Απαραίτητα αρχεία του jyputer book 107](#_Toc180582779)

[Σχήμα 58: \_config.yml 109](#_Toc180582780)

[Σχήμα 59: \_toc.yml 110](#_Toc180582781)

[Σχήμα 60: Εκτελέσιμο αρχείο κώδικα 110](#_Toc180582782)

Κεφάλαιο 1

# Εισαγωγή

## 1.1 Στόχοι της Εργασίας

Η ψηφιακή μάθηση και η διαδραστική εκπαίδευση έχουν αναδειχθεί ως εξαιρετικά αποτελεσματικές μέθοδοι στην κατανόηση πολύπλοκων επιστημονικών αντικειμένων, όπως οι ψηφιακές επικοινωνίες. Η παρούσα διπλωματική εργασία αξιοποιεί αυτές τις καινοτόμες μεθόδους για να διευκολύνει τη διδασκαλία και την κατανόηση των βασικών αρχών των ψηφιακών επικοινωνιών, προσφέροντας μια διαδραστική πλατφόρμα προσομοίωσης συστημάτων μετάδοσης. Η προσέγγιση αυτή βασίζεται στη χρήση ανοιχτού λογισμικού, που παρέχει στους φοιτητές τη δυνατότητα πειραματισμού και ανάλυσης της απόδοσης των συστημάτων σε πραγματικό χρόνο. Με αυτόν τον τρόπο, η θεωρητική γνώση ενισχύεται από την πρακτική εφαρμογή και τις παρατηρήσεις, προσφέροντας ένα ολοκληρωμένο μαθησιακό περιβάλλον που αναπαράγει τις συνθήκες ενός πραγματικού εργαστηρίου ψηφιακών επικοινωνιών.

Η γλώσσα προγραμματισμού Python, που χρησιμοποιείται για την υλοποίηση της διαδραστικής πλατφόρμας αυτής της εργασίας, αποτελεί έναν σημαντικό πυλώνα στην προσπάθεια διευκόλυνσης της διαδικασίας μάθησης. Η Python έχει καθιερωθεί ως μία από τις πιο δημοφιλείς και ευέλικτες γλώσσες προγραμματισμού, λόγω της απλότητας, της εκτεταμένης βιβλιοθήκης εργαλείων και της ικανότητάς της να υποστηρίζει πολυδιάστατες εφαρμογές, όπως η επεξεργασία δεδομένων, οι μαθηματικοί υπολογισμοί, και η ανάπτυξη διαδραστικών γραφικών περιβαλλόντων.

Η Python, με τη διευρυμένη χρήση βιβλιοθηκών όπως η NumPy, SciPy, Matplotlib και άλλες, επιτρέπει την εύκολη και αποτελεσματική υλοποίηση προσομοιώσεων συστημάτων ψηφιακής επικοινωνίας, ενισχύοντας τη διαδραστικότητα και τη δυνατότητα πειραματισμού. Αυτές οι βιβλιοθήκες καθιστούν δυνατή την εκτέλεση σύνθετων μαθηματικών πράξεων και αναλύσεων, όπως η διαμόρφωση και αποκωδικοποίηση σημάτων, η δημιουργία και απεικόνιση διαγραμμάτων, και η ανάλυση της απόδοσης των συστημάτων, μέσα από ένα περιβάλλον απλό στη χρήση αλλά ταυτόχρονα εξαιρετικά ισχυρό σε δυνατότητες.

## 1.2 Δομή της Εργασίας

Η δομή της εργασίας βασίζεται σε μια συντονισμένη χρήση θεωρητικών εργαλείων και πρακτικών εφαρμογών που επιλέχθηκαν για να παρέχουν μια πλήρη εικόνα του θέματος, συγκεκριμένα της ψηφιακής επεξεργασίας σήματος και των εφαρμογών της στις τηλεπικοινωνίες. Τα εργαλεία που χρησιμοποιήθηκαν, όπως οι γλώσσες προγραμματισμού Python και MATLAB, συνδέονται άμεσα με τους στόχους της εργασίας, οι οποίοι περιλαμβάνουν τη θεωρητική κατανόηση και πρακτική υλοποίηση διαδραστικών ασκήσεων.

Η επιλογή της Python, συγκεκριμένα με βιβλιοθήκες όπως οι NumPy, SciPy και Matplotlib, ενισχύει την πρακτική πτυχή της εργασίας, διευκολύνοντας την προσομοίωση των συστημάτων ψηφιακής επικοινωνίας. Οι ασκήσεις προσαρμόστηκαν από το αρχικό περιβάλλον MATLAB στην Python για μεγαλύτερη ευελιξία και αλληλεπίδραση, δημιουργώντας ένα περιβάλλον πιο προσβάσιμο για τους φοιτητές και ενισχύοντας τη μάθηση μέσω πρακτικών παραδειγμάτων και πραγματικών εφαρμογών.

Η δομή της εργασίας αντανακλά αυτή τη σύνδεση θεωρίας και πράξης. Τα αρχικά κεφάλαια εστιάζουν στη θεωρητική ανάλυση των βασικών αρχών της ψηφιακής επεξεργασίας σήματος και των τηλεπικοινωνιακών συστημάτων, ενώ τα μεσαία κεφάλαια επικεντρώνονται στην παρουσίαση και ανάλυση των εργαστηριακών ασκήσεων. Κάθε κεφάλαιο ακολουθεί έναν συγκεκριμένο ρυθμό, ο οποίος ξεκινά με θεωρητικά πλαίσια και συνεχίζει με την εφαρμογή αυτών των θεωριών στην πράξη μέσω της υλοποίησης προσομοιώσεων και πειραματικών δεδομένων.

Συνεπώς, η τελική δομή της εργασίας αποτελεί μια συνεκτική σύνδεση μεταξύ θεωρητικών εννοιών και πρακτικών εργαλείων, ενσωματώνοντας τις καλύτερες πρακτικές από τα δύο πεδία για να παραχθεί ένα εκπαιδευτικό και επιστημονικά τεκμηριωμένο αποτέλεσμα

Κεφάλαιο 2

# Ανασκόπηση της Βιβλιογραφίας

## 2.1 Περιγραφή του μαθήματος

* Εισαγωγή - Η ψηφιακή επεξεργασία σήματος στις τηλεπικοινωνίες – εξομοίωση αναλογικών διαμορφώσεων
* Βέλτιστη ψηφιακή αναγνώριση - Προσαρμοσμένα φίλτρα
* Φασματικά χαρακτηριστικά ψηφιακών κυματομορφών - Μορφοποίηση με φίλτρα Nyquist
* Ψηφιακή Διαμόρφωση QAM και PSK
* Ψηφιακή διαμόρφωση FSK και MSK
* Απόδοση διαύλου – θεωρητικά όρια και πρακτικές προσεγγίσεις
* Ειδικά θέματα και παραδείγματα σύγχρονων συστημάτων ψηφιακής επικοινωνίας [Διαμορφώσεις DMT και OFDM. Συστήματα μετάδοσης DSL και Ψηφιακής Τηλεόρασης DVB-T]

### 2.1.1 Lab Exercise 1: Εισαγωγή στα Συστήματα Ψηφιακής Μετάδοσης

2.1.1.1 Στόχος της πειραματικής άσκησης

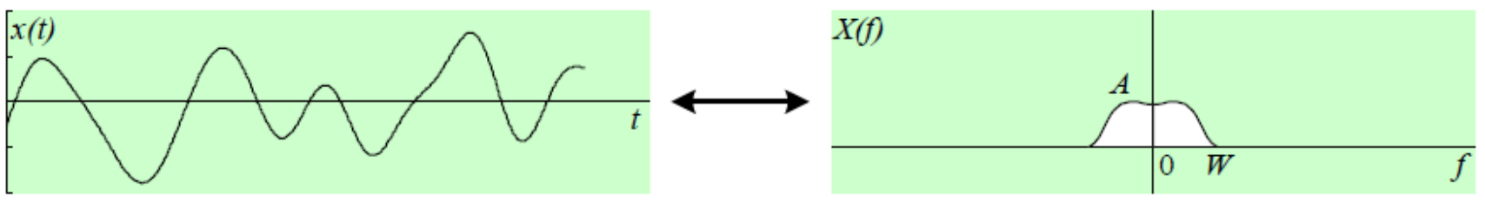
Ο σκοπός του πρώτου συνόλου ασκήσεων είναι, αφενός, η εξοικείωση με το προγραμματιστικό περιβάλλον της Python και της Matlab, και αφετέρου, η εισαγωγή στις μεθόδους αναπαράστασης και επεξεργασίας τηλεπικοινωνιακών σημάτων σε αυτές τις συγκεκριμένες γλώσσες προγραμματισμού.

2.1.1.2 Ανασκόπηση Θεωρίας  
**Δειγματοληψία και Ψηφιοποίηση**

Η δειγματοληψία είναι η διαδικασία με την οποία λαμβάνονται διακριτές τιμές από ένα σήμα συνεχούς χρόνου σε τακτά χρονικά διαστήματα. Σύμφωνα με το θεώρημα της δειγματοληψίας του Nyquist-Shannon, η συχνότητα δειγματοληψίας πρέπει να είναι τουλάχιστον διπλάσια από την ανώτερη συχνότητα του σήματος (fs ≥ 2W) για να αποφευχθεί το φαινόμενο της αναδίπλωσης του φάσματος (aliasing). Αν η συχνότητα δειγματοληψίας είναι χαμηλότερη από αυτό το όριο, τότε εμφανίζονται φασματικά είδωλα που καθιστούν αδύνατη την πλήρη αποκατάσταση του αρχικού σήματος από τα δειγματοληπτημένα δεδομένα.

**Μετασχηματισμός Fourier**

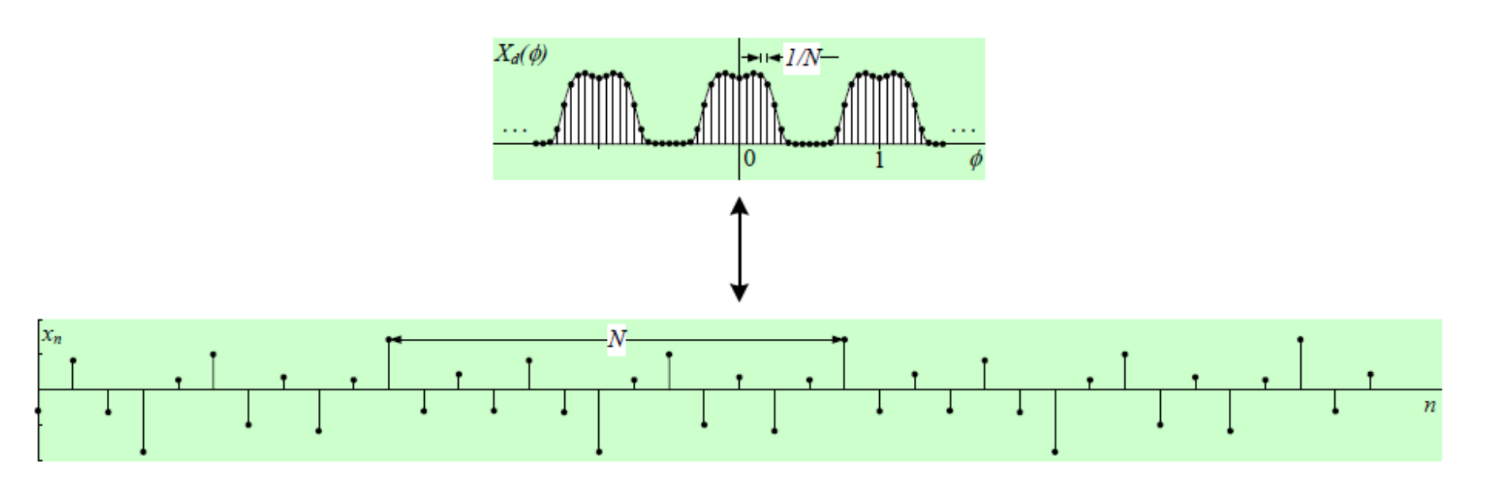
Ο μετασχηματισμός Fourier είναι ένα σημαντικό εργαλείο για την ανάλυση των σημάτων στο πεδίο της συχνότητας. Ο μετασχηματισμός Fourier συνεχούς χρόνου (CTFT) αναλύει ένα σήμα συνεχούς χρόνου σε συχνότητες, επιτρέποντας την εξαγωγή πληροφοριών για το φάσμα του σήματος. Στην περίπτωση διακριτών σημάτων που προκύπτουν από τη δειγματοληψία, χρησιμοποιείται ο μετασχηματισμός Fourier διακριτού χρόνου (DTFT), ο οποίος προσδιορίζει το φάσμα των δειγμάτων στο πεδίο της συχνότητας.



Σχήμα 1: Δειγματοληψία - Ψηφιοποίηση (Μετασχηματισμός Fourier)

Για πρακτικούς λόγους, ο διακριτός μετασχηματισμός Fourier (DFT) είναι αυτός που εφαρμόζεται για την ανάλυση σημάτων πεπερασμένου μήκους. Ο DFT εκτελεί τον ίδιο υπολογισμό όπως ο DTFT, αλλά σε ένα περιορισμένο σύνολο δειγμάτων, παρέχοντας την ανάλυση συχνοτήτων σε ένα πεπερασμένο και συγκεκριμένο αριθμό σημείων.

Σχήμα 3:Δειγματοληψία - Ψηφιοποίηση (Κρουστικές)



Σχήμα 4:Δειγματοληψία - Ψηφιοποίηση (Πεπερασμένη σειρά xn)

**Ενέργεια και Ισχύς Σήματος**

Η ενέργεια και η ισχύς ενός σήματος μπορούν να προσδιοριστούν μέσω της φασματικής ανάλυσης. Η ενέργεια του σήματος είναι το ολοκλήρωμα του τετραγώνου του σήματος στο πεδίο του χρόνου, ενώ η ισχύς ενός περιοδικού σήματος είναι η μέση ενέργεια ανά περίοδο. Στην περίπτωση σημάτων διακριτού χρόνου, η ενέργεια υπολογίζεται ως το άθροισμα των τετραγώνων των δειγμάτων του σήματος.

Μια σημαντική μέθοδος εκτίμησης της ισχύος ενός σήματος είναι η χρήση του περιοδογράμματος, το οποίο βασίζεται στον μετασχηματισμό Fourier του σήματος και παρέχει μια εκτίμηση της πυκνότητας φάσματος ισχύος (PSD). Το περιοδόγραμμα επιτρέπει την ανάλυση της κατανομής της ισχύος του σήματος στις διάφορες συχνότητες και αποτελεί βασικό εργαλείο για τη φασματική ανάλυση σημάτων.

**Σφάλματα και Αναδίπλωση (Aliasing)**

Όπως προαναφέρθηκε, εάν η δειγματοληψία ενός σήματος πραγματοποιηθεί με συχνότητα μικρότερη από τη συχνότητα Nyquist, το αποτέλεσμα είναι η εμφάνιση φασματικών ειδώλων και η απώλεια της ακρίβειας στην ανακατασκευή του σήματος. Το φαινόμενο αυτό ονομάζεται αναδίπλωση (aliasing) και αποτελεί μια από τις κύριες προκλήσεις στη δειγματοληψία αναλογικών σημάτων. Το σφάλμα αναδίπλωσης προκαλεί παραμορφώσεις που δεν μπορούν να διορθωθούν, καθιστώντας αναγκαία τη χρήση φίλτρων για την αποφυγή του.

**Εφαρμογή στην Πράξη**

Στο πλαίσιο της εργαστηριακής άσκησης, οι παραπάνω θεωρητικές έννοιες εφαρμόζονται για την ανάλυση ενός ημιτονοειδούς σήματος, τη δειγματοληψία του, την εκτέλεση του διακριτού μετασχηματισμού Fourier, και την εξαγωγή του περιοδογράμματος του σήματος. Ο υπολογισμός της ισχύος και η φασματική ανάλυση παρέχουν μια ολοκληρωμένη κατανόηση της συμπεριφοράς του σήματος και των χαρακτηριστικών του στο πεδίο του χρόνου και της συχνότητας.

### 2.1.2 Lab Exercise 2: Ψηφιακά Φίλτρα

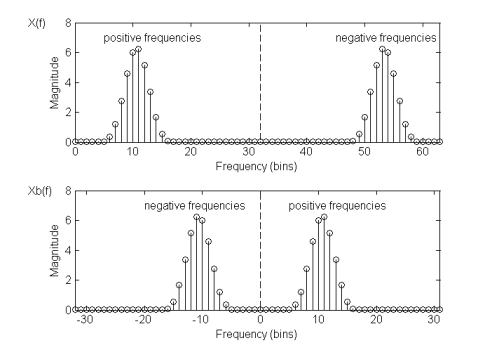
2.1.2.1 Στόχος της πειραματικής άσκησης

Σκοπός της δεύτερης σειράς ασκήσεων είναι η εξοικείωση με τις συναρτήσεις σχεδιασμού φίλτρων πεπερασμένης κρουστικής απόκρισης (FIR) και την υλοποίησή τους σε Python. Προτού ξεκινήσετε την άσκηση θα πρέπει να μελετήσετε με προσοχή το Κεφάλαιο 1 και, ειδικότερα, την παράγραφο 1.3 του τεύχους του μαθήματος

2.1.2.2 Ανασκόπηση Θεωρίας

**Ψηφιακά Φίλτρα**

Τα ψηφιακά φίλτρα FIR είναι ένας τύπος φίλτρων που χρησιμοποιούνται ευρέως στην επεξεργασία ψηφιακών σημάτων λόγω της σταθερότητάς τους και της προβλέψιμης απόκρισης τους. Σε αντίθεση με τα φίλτρα άπειρης κρουστικής απόκρισης (IIR), τα FIR φίλτρα δεν εμφανίζουν αναδράσεις, κάτι που εξασφαλίζει ότι η έξοδος ενός FIR φίλτρου εξαρτάται μόνο από ένα πεπερασμένο αριθμό δειγμάτων του εισερχόμενου σήματος.



Σχήμα : Αμφίπλευρη και μονόπλευρη αναπαράσταση του μετασχηματισμού Fourier του ημιρονοειδούς σήματος

Το FIR φίλτρο ορίζεται από μια ακολουθία συντελεστών, η οποία αντιστοιχεί στην κρουστική απόκριση του φίλτρου. Η κρουστική απόκριση υπολογίζεται ως ο αντίστροφος μετασχηματισμός Fourier της ιδανικής απόκρισης συχνότητας. Στην πράξη, η ακριβής εφαρμογή της κρουστικής απόκρισης περιορίζεται σε ένα πεπερασμένο αριθμό δειγμάτων, οδηγώντας σε σφάλματα στις άκρες του φάσματος, τα οποία όμως μπορούν να μειωθούν με τη χρήση κατάλληλων παραθύρων.

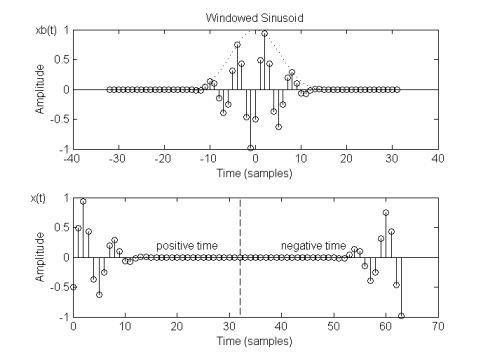
**Σχεδιασμός Φίλτρων**

Στην άσκηση, εξετάζονται δύο βασικές μέθοδοι σχεδιασμού FIR φίλτρων:

1. Μέθοδος των Παραθύρων: Η μέθοδος αυτή χρησιμοποιεί διαφορετικά παράθυρα (π.χ. παράθυρο Hamming ή Kaiser) για τον περιορισμό της κρουστικής απόκρισης. Το αποτέλεσμα αυτής της μεθόδου είναι η μείωση των πλευρικών λοβών, οι οποίοι προκύπτουν από τη διακοπή της κρουστικής απόκρισης.
2. Μέθοδος Parks-McClellan: Πρόκειται για μια πιο εξειδικευμένη μέθοδο σχεδιασμού, η οποία χρησιμοποιεί την τεχνική των ισοϋψών κυματώσεων. Με την τεχνική αυτή επιτυγχάνεται ένας βέλτιστος σχεδιασμός του φίλτρου, με έμφαση στη βελτιστοποίηση της απόκρισης συχνότητας στις ζώνες διέλευσης και αποκοπής.

**Μετασχηματισμοί Fourier και Ανάλυση Συχνότητας**

Η κατανόηση των μετασχηματισμών Fourier είναι κρίσιμη για την ανάλυση σημάτων. Ο μετασχηματισμός Fourier ενός σήματος επιτρέπει την ανάλυση του στο πεδίο της συχνότητας, αποκαλύπτοντας τις συχνότητες που συνθέτουν το σήμα. Στην άσκηση, χρησιμοποιούνται οι συναρτήσεις fft και ifft για τον υπολογισμό του μετασχηματισμού Fourier και του αντίστροφου μετασχηματισμού αντίστοιχα. Η αναπαράσταση ενός σήματος στο πεδίο της συχνότητας είναι κρίσιμη για την ανάλυση της απόκρισης των φίλτρων και για την κατανόηση του τρόπου με τον οποίο το φίλτρο επηρεάζει τα συχνιακά χαρακτηριστικά του σήματος.



Σχήμα 5: Αμφίπλευρη και μονόπλευρη αναπαράσταση ενός ημιτονοειδούς σήματος που έχει πολλαπλασιαστεί με παράθυρο Blackman

**Σφάλματα και Ιδιομορφίες στις Αναπαραστάσεις Συχνοτήτων**

Κατά τη διάρκεια της εφαρμογής των μετασχηματισμών Fourier, προκύπτουν ιδιομορφίες όπως η αμφίπλευρη και η μονόπλευρη αναπαράσταση των συχνοτήτων, όπου οι αρνητικές συχνότητες αναπαρίστανται στο άνω μισό του φάσματος. Η χρήση συναρτήσεων όπως η fftshift και ifftshift επιτρέπει την κυκλική μετατόπιση των συχνοτήτων ώστε να κεντραριστούν στο μηδέν, κάτι που διευκολύνει την ερμηνεία της φασματικής ανάλυσης.

**Εφαρμογές Φίλτρων και Ανάλυση Αποτελεσμάτων**

Η άσκηση ολοκληρώνεται με την εφαρμογή των σχεδιασμένων φίλτρων σε πραγματικά σήματα, όπως το σήμα sonar, το οποίο χρησιμοποιείται για να επιδειχθούν οι διαφορές μεταξύ των φίλτρων. Η σύγκριση των διαφορετικών παραθύρων και μεθόδων σχεδιασμού γίνεται με βάση την απόκριση συχνότητας του φίλτρου και την επίδρασή τους στο φάσμα ισχύος του φιλτραρισμένου σήματος. Ο εκτιμητής Welch χρησιμοποιείται για τον υπολογισμό της πυκνότητας φάσματος ισχύος, προσφέροντας μια συνολική εικόνα της φασματικής κατανομής του σήματος μετά την εφαρμογή των φίλτρων.

### 2.1.3 Lab Exercise 3: Προσαρμοσμένα Φίλτρα και Διαμόρφωση L-ASK

2.1.3.1 Στόχος της πειραματικής άσκησης

Ο σκοπός της τρίτης εργαστηριακής άσκησης είναι η μελέτη και διερεύνηση των προσαρμοσμένων φίλτρων και της πολυεπίπεδης διαμόρφωσης L-ASK (Amplitude Shift Keying), μέσω της υλοποίησης και εξομοίωσης της διαδικασίας μετάδοσης και ανίχνευσης σημάτων. Ειδικότερα, εστιάζει στην ανάλυση της επίδοσης των φίλτρων προσαρμογής σε θορυβώδη περιβάλλοντα, καθώς και στη διερεύνηση της πιθανότητας λάθους κατά τη μετάδοση συμβόλων, συνεισφέροντας στην κατανόηση βασικών αρχών των ψηφιακών επικοινωνιών και της αξιολόγησης των επικοινωνιακών συστημάτων σε όρους σηματοθορυβικού λόγου (SNR) και Bit Error Rate (BER).

2.1.3.2 Ανασκόπηση Θεωρίας

**Προσαρμοσμένα Φίλτρα**

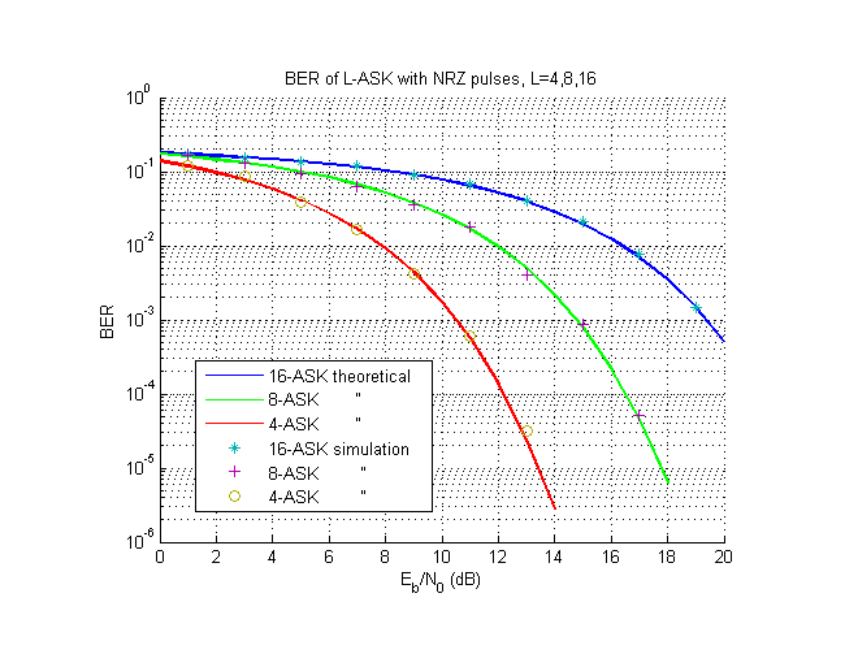
Τα προσαρμοσμένα φίλτρα (matched filters) είναι ειδικοί τύποι ψηφιακών φίλτρων που χρησιμοποιούνται για την ανίχνευση ενός γνωστού σήματος το οποίο μεταδίδεται σε θορυβώδες περιβάλλον. Το προσαρμοσμένο φίλτρο είναι σχεδιασμένο με τέτοιο τρόπο ώστε να μεγιστοποιεί τον λόγο σήματος προς θόρυβο (SNR) στη στιγμή ανίχνευσης του σήματος, επιτρέποντας την καλύτερη ανίχνευση των συμβόλων που μεταδίδονται μέσω του θορυβώδους καναλιού.

Στη συγκεκριμένη άσκηση, το προσαρμοσμένο φίλτρο εφαρμόζεται σε μια ακολουθία υπερδειγματοληπτημένων σημάτων, όπου μετά την επεξεργασία, η έξοδος του φίλτρου χρησιμοποιείται για τη λήψη των συμβόλων. Αυτό επιτυγχάνεται μέσω της συνέλιξης του σήματος με το προσαρμοσμένο φίλτρο, το οποίο ουσιαστικά αντιστρέφει τη χρονική σειρά του αρχικού παλμού του σήματος.

**Διαμόρφωση L-ASK**

Η πολυεπίπεδη διαμόρφωση L-ASK (Amplitude Shift Keying) είναι μια τεχνική διαμόρφωσης όπου τα δεδομένα κωδικοποιούνται σε διαφορετικά επίπεδα πλάτους του φέροντος σήματος. Κάθε επίπεδο πλάτους αντιστοιχεί σε ένα σύμβολο δεδομένων, και επομένως η διαμόρφωση ASK επιτρέπει την κωδικοποίηση πολλαπλών bits σε κάθε σύμβολο. Στην άσκηση, η L-ASK εφαρμόζεται για την ανάλυση της επίδοσης του συστήματος σε διαφορετικές τιμές του σηματοθορυβικού λόγου (SNR), και η πιθανότητα λάθους (BER) υπολογίζεται για διαφορετικά επίπεδα θορύβου.

Η διαδικασία αυτή περιλαμβάνει την εξομοίωση της μετάδοσης και ανίχνευσης σημάτων σε διάφορα επίπεδα θορύβου. Ο ρόλος του προσαρμοσμένου φίλτρου είναι να φιλτράρει το θορυβώδες σήμα, διευκολύνοντας τη σωστή λήψη των συμβόλων. Στη συνέχεια, η διαδικασία ανίχνευσης ελαχιστοποιεί τη διαφορά μεταξύ του λαμβανόμενου σήματος και των προεπιλεγμένων επιπέδων πλάτους, κάνοντας απόφαση για το πιο κοντινό επίπεδο και καταμετρώντας τα σφάλματα στη λήψη.



Σχήμα 7: BER συναρτήση του σγματοθορυβικού λόγου

**Σηματοθορυβικός Λόγος (SNR) και Bit Error Rate (BER)**

Η επίδοση των ψηφιακών επικοινωνιακών συστημάτων αξιολογείται συχνά με βάση την πιθανότητα λάθους κατά τη μετάδοση (Bit Error Rate - BER) σε σχέση με τον σηματοθορυβικό λόγο (SNR). Η εξομοίωση σε αυτή την άσκηση περιλαμβάνει τον υπολογισμό της BER για διαφορετικές τιμές του SNR και την οπτική απεικόνιση της σχέσης μεταξύ τους μέσω καμπυλών απόδοσης.

Συγκεκριμένα, η πιθανότητα λάθους σε συστήματα διαμόρφωσης ASK εξαρτάται από τον αριθμό των επιπέδων (L) και τον σηματοθορυβικό λόγο. Καθώς αυξάνεται το L, αυξάνεται και η πολυπλοκότητα της ανίχνευσης, κάτι που οδηγεί σε αύξηση της πιθανότητας λάθους σε θορυβώδη περιβάλλοντα, εκτός εάν ο SNR είναι επαρκώς υψηλός.

### 2.1.4 Lab Exercise 4: Σηματοδότηση Nyquist και Διαμόρφωση L-ASK

2.1.4.1 Στόχος της πειραματικής άσκησης

Ο σκοπός της τέταρτης εργαστηριακής άσκησης είναι η διερεύνηση της απόδοσης και της εφαρμογής των φίλτρων Nyquist σε συστήματα ψηφιακής επικοινωνίας. Μέσα από την προσομοίωση και ανάλυση διαμορφώσεων, όπως η 8-ASK, οι φοιτητές καλούνται να κατανοήσουν τις βασικές αρχές της σηματοδοσίας Nyquist, την επίδραση του roll-off στα φίλτρα, και την επίδοση του συστήματος σε όρους Bit Error Rate (BER) συναρτήσει του λόγου Eb/No. Η άσκηση περιλαμβάνει την πρακτική υλοποίηση φίλτρων Nyquist και την ανάλυση της απόδοσής τους μέσω γραφικών απεικονίσεων, προσφέροντας μια ολοκληρωμένη κατανόηση των θεωρητικών και πρακτικών πτυχών της σηματοδοσίας Nyquist​.

2.1.4.2 Ανασκόπηση Θεωρίας

**Σηματοδοσία Nyquist**

Η σηματοδοσία Nyquist αποτελεί μία μέθοδο με την οποία μπορούμε να εξασφαλίσουμε ότι τα διαδοχικά σύμβολα που μεταδίδονται δεν παρεμβάλλονται μεταξύ τους, μειώνοντας την πιθανότητα σφαλμάτων κατά την ανίχνευση των συμβόλων στον δέκτη. Αυτό επιτυγχάνεται μέσω της χρήσης των φίλτρων Nyquist, τα οποία έχουν συγκεκριμένα χαρακτηριστικά που εξασφαλίζουν ότι το σήμα λαμβάνει τη μέγιστη τιμή του μόνο στη διάρκεια της περιόδου συμβόλων και μηδενίζεται στα υπόλοιπα σημεία.

Μια βασική παράμετρος στα φίλτρα Nyquist είναι ο συντελεστής roll-off (α), ο οποίος καθορίζει το εύρος του φάσματος συχνοτήτων του φίλτρου. Μικρότερες τιμές του α αυξάνουν την πυκνότητα φάσματος, αλλά οδηγούν σε μεγαλύτερη πολυπλοκότητα στον σχεδιασμό του φίλτρου και στην ευαισθησία σε σφάλματα, ενώ μεγαλύτερες τιμές του α διευκολύνουν τη σταθερότητα του συστήματος με την επέκταση του φάσματος συχνοτήτων.

**Διαμόρφωση L-ASK**

Η διαμόρφωση L-ASK είναι μια τεχνική όπου τα δεδομένα κωδικοποιούνται σε επίπεδα πλάτους του σήματος. Σε κάθε σύμβολο αντιστοιχεί ένα συγκεκριμένο επίπεδο πλάτους, επιτρέποντας έτσι την αποστολή πολλαπλών bits ανά σύμβολο. Για παράδειγμα, στη διαμόρφωση 8-ASK, έχουμε οκτώ επίπεδα πλάτους και επομένως τρία bits μπορούν να μεταδοθούν σε κάθε σύμβολο. Η κωδικοποίηση Gray χρησιμοποιείται συχνά για να ελαχιστοποιήσει τα σφάλματα, εξασφαλίζοντας ότι η μετάβαση από το ένα επίπεδο πλάτους στο επόμενο αντιστοιχεί σε αλλαγή ενός μόνο bit.

**Επίδραση του Roll-off στα Φίλτρα**

Το roll-off των φίλτρων Nyquist επηρεάζει σημαντικά την απόδοση του συστήματος. Όσο μικρότερο είναι το roll-off, τόσο πιο απότομα είναι τα όρια του φάσματος συχνοτήτων, κάτι που μειώνει το εύρος ζώνης που καταλαμβάνει το σήμα. Ωστόσο, αυτό μπορεί να οδηγήσει σε μεγαλύτερη πολυπλοκότητα στην κατασκευή του φίλτρου. Αντίθετα, ένα μεγαλύτερο roll-off εξασφαλίζει πιο ήπια μετάβαση στις συχνότητες αλλά καταλαμβάνει μεγαλύτερο εύρος ζώνης.

**Απόδοση Συστήματος και BER**

Η απόδοση του συστήματος αξιολογείται μέσω της πιθανότητας λάθους κατά τη μετάδοση, δηλαδή του Bit Error Rate (BER). Το BER σχετίζεται άμεσα με τον σηματοθορυβικό λόγο Eb/No και καθορίζει την ποιότητα της επικοινωνίας. Καθώς αυξάνεται το Eb/No, μειώνεται η πιθανότητα λάθους, βελτιώνοντας την απόδοση του συστήματος.

Στην άσκηση, καλείστε να υπολογίσετε και να αναλύσετε την καμπύλη BER σε συνάρτηση με το Eb/No για διαφορετικές παραμέτρους του φίλτρου, όπως ο συντελεστής roll-off και η τάξη του φίλτρου. Οι διαφορετικές τιμές roll-off θα σας επιτρέψουν να κατανοήσετε την επίδραση του εύρους ζώνης και της πολυπλοκότητας στην απόδοση του συστήματος.

**Εφαρμογή της Σηματοδοσίας Nyquist στην Πράξη**

Η σηματοδοσία Nyquist, σε συνδυασμό με τη διαμόρφωση L-ASK, χρησιμοποιείται ευρέως σε συστήματα ψηφιακών επικοινωνιών που απαιτούν υψηλό ρυθμό μετάδοσης δεδομένων, διατηρώντας παράλληλα ένα αποδεκτό επίπεδο σφαλμάτων. Στην πράξη, η σηματοδοσία αυτή επιτρέπει την αύξηση του ρυθμού μετάδοσης χωρίς να απαιτείται υπερβολικό εύρος ζώνης, κάτι που την καθιστά ιδανική για εφαρμογές όπως η ψηφιακή τηλεόραση, τα δίκτυα κινητής τηλεφωνίας και τα δορυφορικά συστήματα επικοινωνίας.

### 2.1.5 Lab Exercise 5: Διαμόρφωση QAM και PSK

2.1.5.1 Στόχος της πειραματικής άσκησης

Ο σκοπός της πέμπτης εργαστηριακής άσκησης είναι η μελέτη και ανάλυση των συστημάτων διαμόρφωσης QAM και PSK, καθώς και η διερεύνηση της απόδοσής τους σε διαφορετικές συνθήκες σηματοθορυβικού λόγου (Eb/No). Οι φοιτητές καλούνται να σχεδιάσουν σηματικούς αστερισμούς, να προσομοιώσουν πομπούς και δέκτες για διαφορετικές τιμές της παραμέτρου roll-off και να αναλύσουν την επίδοση των συστημάτων μέσω της καμπύλης Bit Error Rate (BER). Η άσκηση στοχεύει στην κατανόηση των παραμέτρων που επηρεάζουν την απόδοση των συστημάτων διαμόρφωσης και την επίτευξη του βέλτιστου ρυθμού μετάδοσης δεδομένων​.

2.1.5.2 Ανασκόπηση Θεωρίας

**Διαμόρφωση QAM**

Η QAM είναι μια διαμόρφωση όπου τόσο το πλάτος όσο και η φάση του φέροντος σήματος τροποποιούνται ταυτόχρονα για να κωδικοποιηθούν τα δεδομένα. Αυτό επιτρέπει τη μετάδοση μεγάλου αριθμού bits ανά σύμβολο, με τη χρήση ενός σηματικού αστερισμού ορθογωνικού πλέγματος, όπου κάθε σημείο αντιστοιχεί σε έναν συνδυασμό φάσης και πλάτους.

Στο πλαίσιο της άσκησης, οι φοιτητές καλούνται να μελετήσουν τη διαμόρφωση 64-QAM, η οποία χρησιμοποιεί 64 διαφορετικά σημεία στον σηματικό αστερισμό για να κωδικοποιήσει έξι bits ανά σύμβολο. Επιπλέον, η κωδικοποίηση Gray εφαρμόζεται για να μειωθεί η πιθανότητα λάθους, εξασφαλίζοντας ότι γειτονικά σημεία του αστερισμού διαφέρουν μόνο κατά ένα bit.

Η QAM προσφέρει υψηλότερη αποδοτικότητα φάσματος σε σχέση με άλλες διαμορφώσεις, καθώς επιτρέπει τη μετάδοση μεγάλου όγκου δεδομένων χωρίς να απαιτείται υπερβολικό εύρος ζώνης. Ωστόσο, είναι πιο ευαίσθητη στον θόρυβο και στις παραμορφώσεις του καναλιού, κάτι που καθιστά αναγκαία τη διατήρηση υψηλών επιπέδων σηματοθορυβικού λόγου (Eb/No) για να εξασφαλιστεί χαμηλή πιθανότητα σφάλματος (BER).

**Διαμόρφωση PSK**

Η PSK είναι μια διαμόρφωση όπου η πληροφορία κωδικοποιείται αποκλειστικά στη φάση του φέροντος σήματος, ενώ το πλάτος παραμένει σταθερό. Η απλούστερη μορφή είναι η BPSK (Binary PSK), όπου η φάση του σήματος λαμβάνει δύο τιμές, αντιστοιχώντας σε δύο διαφορετικά bits. Για μεγαλύτερους ρυθμούς μετάδοσης, χρησιμοποιείται η QPSK (Quadrature PSK) ή άλλες υψηλότερης τάξης παραλλαγές της PSK, όπου πολλαπλά bits κωδικοποιούνται σε περισσότερες φάσεις.

Η PSK, σε σύγκριση με την QAM, είναι πιο ανθεκτική στον θόρυβο και στις διαταραχές, καθώς η πληροφορία εξαρτάται μόνο από τη φάση του σήματος. Ωστόσο, η αποδοτικότητά της σε όρους εύρους ζώνης είναι μικρότερη από την QAM, καθώς απαιτεί μεγαλύτερο εύρος ζώνης για την επίτευξη του ίδιου ρυθμού μετάδοσης δεδομένων.

**Σηματικοί Αστερισμοί και Κωδικοποίηση Gray**

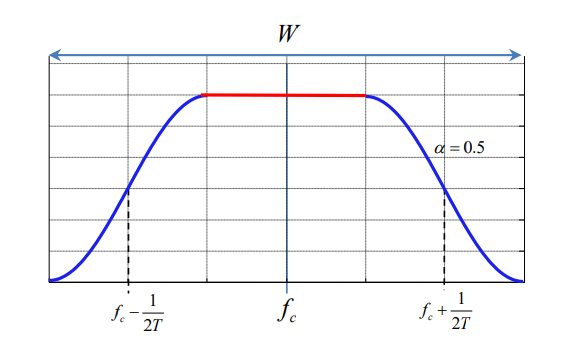
Η σημασία των σηματικών αστερισμών είναι καθοριστική για την κατανόηση των συστημάτων διαμόρφωσης QAM και PSK. Οι αστερισμοί αντιπροσωπεύουν τα πιθανά σημεία που μπορεί να λάβει το σήμα κατά τη μετάδοση, με κάθε σημείο να αντιστοιχεί σε έναν συγκεκριμένο συνδυασμό bits. Η κωδικοποίηση Gray εξασφαλίζει ότι γειτονικά σημεία διαφέρουν μόνο κατά ένα bit, μειώνοντας έτσι την πιθανότητα σφάλματος όταν το σήμα επηρεάζεται από θόρυβο ή παρεμβολές.

**Σηματοθορυβικός Λόγος (Eb/No) και Bit Error Rate (BER)**

Η καμπύλη BER σε συνάρτηση με το Eb/No είναι ένα βασικό εργαλείο για την αξιολόγηση της απόδοσης των συστημάτων διαμόρφωσης. Καθώς αυξάνεται το Eb/No, μειώνεται η πιθανότητα λάθους, οδηγώντας σε πιο αξιόπιστη μετάδοση των δεδομένων. Η άσκηση επικεντρώνεται στη μελέτη των καμπυλών αυτών τόσο για την QAM όσο και για την PSK, δίνοντας έμφαση στις διαφορές που παρουσιάζουν ανάλογα με τις συνθήκες του καναλιού και τις παραμέτρους διαμόρφωσης.

**Ρυθμός Μετάδοσης και Αποδοτικότητα Εύρους Ζώνης**

Ο ρυθμός μετάδοσης καθορίζεται από την ποσότητα των bits που μπορούν να μεταδοθούν ανά σύμβολο και τον αριθμό των συμβόλων ανά δευτερόλεπτο (Baud Rate). Συστήματα με μεγαλύτερους σηματικούς αστερισμούς, όπως η 64-QAM, μπορούν να μεταδώσουν περισσότερα bits ανά σύμβολο, αυξάνοντας τον συνολικό ρυθμό μετάδοσης. Ωστόσο, αυτό απαιτεί υψηλότερο Eb/No για να επιτευχθεί χαμηλό BER.



Σχήμα : Φάσμα φίλτρου με εύρος ζώνης W γύρω από τη συχνότητα φέροντος fc

Επίσης, η χρήση του φίλτρου Nyquist με κατάλληλο roll-off factor επηρεάζει το απαιτούμενο εύρος ζώνης του συστήματος. Το roll-off factor καθορίζει πόσο γρήγορα μειώνεται το φάσμα του σήματος και παίζει κρίσιμο ρόλο στη βελτιστοποίηση του εύρους ζώνης.

### 2.1.6 Lab Exercise 6: Διαμόρφωση FSK και MSK

2.1.6.1 Στόχος της πειραματικής άσκησης

Ο σκοπός αυτής της εργαστηριακής άσκησης είναι η ανάλυση και σύγκριση των τεχνικών διαμόρφωσης FSK (Frequency Shift Keying) και MSK (Minimum Shift Keying), μέσω προσομοιώσεων και θεωρητικών υπολογισμών. Η άσκηση επικεντρώνεται στην εξέταση της απόδοσης των συστημάτων αυτών, κυρίως σε όρους Bit Error Rate (BER) και εύρους ζώνης, τόσο για σύμφωνη όσο και για ασύμφωνη αποδιαμόρφωση. Επιπλέον, οι φοιτητές καλούνται να συγκρίνουν το MSK με το ισοδύναμο σύστημα QPSK, παρέχοντας μια ολοκληρωμένη κατανόηση της επίδρασης της διαμόρφωσης στην απόδοση των επικοινωνιακών συστημάτων.

2.1.6.2 Ανασκόπηση Θεωρίας

**Διαμόρφωση FSK (Frequency Shift Keying)**

Η FSK είναι μια μέθοδος διαμόρφωσης όπου η πληροφορία κωδικοποιείται μέσω της αλλαγής της συχνότητας του φέροντος σήματος. Για παράδειγμα, στο δυαδικό FSK (2-FSK), δύο διαφορετικές συχνότητες αντιστοιχούν σε δύο λογικές καταστάσεις (0 και 1). Συστήματα FSK υψηλότερης τάξης, όπως το 16-FSK ή το 32-FSK, χρησιμοποιούν πολλαπλές συχνότητες για να κωδικοποιήσουν περισσότερα bits ανά σύμβολο.

Η απόδοση της FSK διαμόρφωσης εξαρτάται από τον τρόπο αποδιαμόρφωσης. Υπάρχουν δύο κύριες μέθοδοι:

1. Σύμφωνη αποδιαμόρφωση (coherent detection), όπου ο δέκτης έχει πληροφορίες σχετικά με τη φάση του φέροντος σήματος, εξασφαλίζοντας πιο αξιόπιστη αποδιαμόρφωση, αλλά απαιτεί μεγαλύτερη πολυπλοκότητα.
2. Ασύμφωνη αποδιαμόρφωση (non-coherent detection), όπου ο δέκτης δεν χρησιμοποιεί πληροφορίες για τη φάση του σήματος, κάτι που καθιστά το σύστημα πιο απλό αλλά με αυξημένη πιθανότητα λάθους (BER).

Η FSK είναι ανθεκτική στον θόρυβο και στις παρεμβολές, καθιστώντας την ιδιαίτερα χρήσιμη σε εφαρμογές όπως ασύρματα δίκτυα και συστήματα ραδιοεπικοινωνιών.

**Διαμόρφωση MSK (Minimum Shift Keying)**

Η MSK είναι μια ειδική περίπτωση της FSK με την ελάχιστη δυνατή απόσταση μεταξύ των συχνοτήτων, εξασφαλίζοντας ότι οι φάσεις των διαδοχικών συμβόλων είναι συνεχείς. Αυτό σημαίνει ότι το σήμα MSK χαρακτηρίζεται από μικρότερη διασπορά του φάσματος σε σύγκριση με την τυπική FSK, επιτρέποντας αποδοτικότερη χρήση του εύρους ζώνης.

Το MSK έχει χαρακτηριστεί ως μια από τις πλέον αποδοτικές διαμορφώσεις σε όρους εύρους ζώνης, ενώ η συνεχής φάση του σήματος οδηγεί σε χαμηλότερη πιθανότητα σφάλματος σε σύγκριση με άλλες διαμορφώσεις. Επιπλέον, η MSK έχει παρόμοια απόδοση με την QPSK (Quadrature Phase Shift Keying), κάτι που την καθιστά ισοδύναμη επιλογή για πολλές εφαρμογές επικοινωνίας.

**Σύγκριση MSK και QPSK**

Η QPSK είναι μια διαμόρφωση όπου τα δεδομένα κωδικοποιούνται σε τέσσερις διαφορετικές φάσεις, με δύο bits ανά σύμβολο. Το MSK, παρόλο που είναι μια συνεχής φασική διαμόρφωση, έχει παρόμοια φασματική απόδοση με την QPSK και μπορεί να χρησιμοποιηθεί για την επίτευξη ίδιων ρυθμών μετάδοσης δεδομένων.

Η σύγκριση μεταξύ MSK και QPSK περιλαμβάνει την ανάλυση του Bit Error Rate (BER) για διαφορετικές τιμές του Eb/No, καθώς και την αποδοτικότητα του εύρους ζώνης. Το MSK, λόγω της συνεχούς φάσης, εμφανίζει καλύτερη απόδοση σε θορυβώδη περιβάλλοντα, ενώ η QPSK μπορεί να είναι πιο ευέλικτη για συστήματα όπου η πολυπλοκότητα του δέκτη δεν είναι ζήτημα.

**Εύρος Ζώνης και Bit Error Rate (BER)**

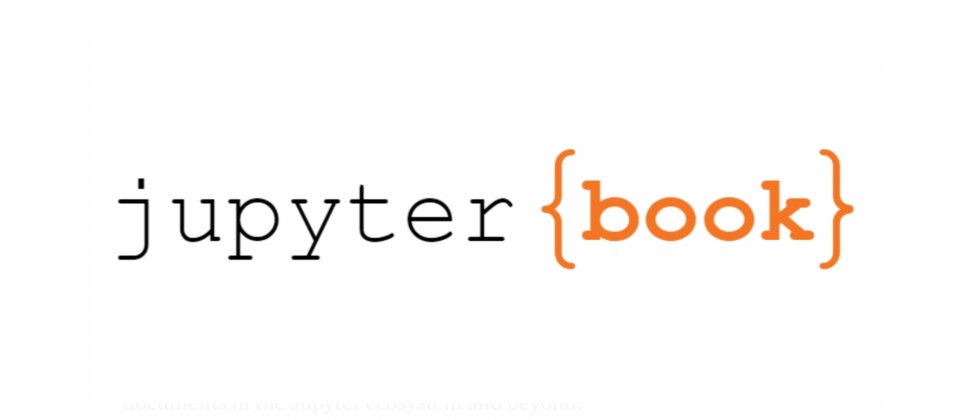
Η απόδοση των συστημάτων FSK και MSK αξιολογείται μέσω της καμπύλης BER σε συνάρτηση με το Eb/No. Καθώς αυξάνεται το Eb/No, μειώνεται η πιθανότητα σφάλματος, βελτιώνοντας την απόδοση του συστήματος. Η σύμφωνη αποδιαμόρφωση προσφέρει χαμηλότερα ποσοστά λάθους σε σχέση με την ασύμφωνη, ειδικά σε υψηλές τιμές Eb/No. Ωστόσο, η ασύμφωνη αποδιαμόρφωση είναι πιο απλή και ενδείκνυται για συστήματα όπου η απλότητα του δέκτη είναι κρίσιμη.

Επιπλέον, το εύρος ζώνης που απαιτείται για τη μετάδοση ενός σήματος εξαρτάται από τη διαμόρφωση και το ρυθμό μετάδοσης. Το MSK είναι πιο αποδοτικό σε όρους εύρους ζώνης σε σύγκριση με άλλες διαμορφώσεις όπως η FSK και η QPSK, γεγονός που το καθιστά προτιμητέα επιλογή σε περιβάλλοντα όπου το διαθέσιμο εύρος ζώνης είναι περιορισμένο.

## 2.2 Εργαλεία που χρησιμοποιήθηκαν

Η διερεύνηση και τα κριτήρια επιλογής των εργαλείων που χρησιμοποιήθηκαν στη διπλωματική εργασία βασίστηκαν στο τεύχος του Αραβαντινού Σωτήρη, όπου έγινε μια εκτενής ανάλυση διαφόρων λογισμικών και βιβλιοθηκών που θα μπορούσαν να συμβάλουν στην επίτευξη των στόχων της μελέτης. Τα εργαλεία που τελικά επιλέξαμε για την υλοποίηση των επιμέρους τμημάτων της εργασίας ήταν τα εξής:

1. Το Jupyter Book είναι μια πλατφόρμα που χρησιμοποιείται για τη δημιουργία διαδραστικών, πλήρως τεκμηριωμένων επιστημονικών βιβλίων. Είναι σχεδιασμένο να υποστηρίζει κώδικα, διαγράμματα, εξισώσεις και διαδραστικά στοιχεία, και συνδυάζει την ευκολία των Jupyter Notebooks με τη δομή και την επαγγελματική παρουσίαση ενός βιβλίου. Σε αυτή την εργασία, το Jupyter Book χρησιμοποιήθηκε για την ανάπτυξη διαδραστικού υλικού, επιτρέποντας στους αναγνώστες να εκτελούν κώδικα, να αλλάζουν παραμέτρους, και να αλληλεπιδρούν με τα μαθηματικά μοντέλα απευθείας μέσα από την πλατφόρμα, διευκολύνοντας την κατανόηση της θεωρίας και των αποτελεσμάτων.

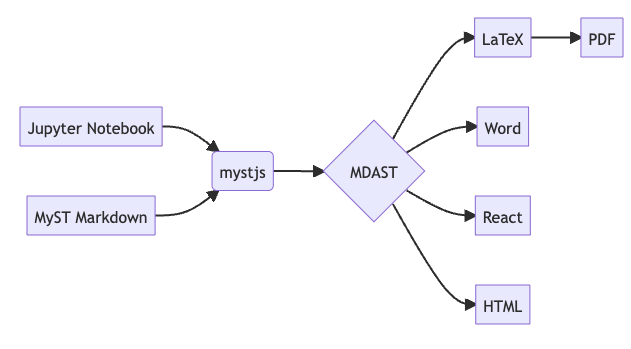


1. Το MyST Markdown (Markedly Structured Text) είναι μια επεκτάσιμη μορφή Markdown που αναπτύχθηκε για χρήση σε τεχνικά και ακαδημαϊκά περιβάλλοντα, όπως τα Jupyter Notebooks και το Jupyter Book. Αποτελεί μια προσαρμοσμένη μορφή της γλώσσας Markdown, η οποία προσφέρει επιπλέον δυνατότητες για τη συγγραφή πιο δομημένου και πλούσιου περιεχομένου σε κείμενα, μαθηματικά σύμβολα και διαδραστικά στοιχεία.



Κύρια Χαρακτηριστικά του MyST Markdown:

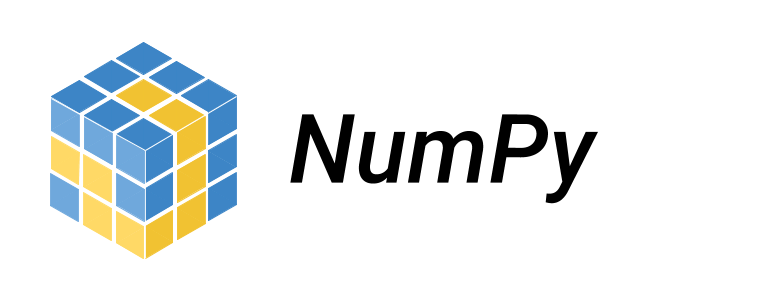
1. Υποστήριξη Jupyter Notebooks: Το MyST Markdown χρησιμοποιείται ευρέως στα Jupyter Book και Jupyter Notebooks. Επιτρέπει την ενσωμάτωση εκτελέσιμου κώδικα και εξόδου από Jupyter Notebooks απευθείας μέσα σε κείμενα Markdown.
2. Επέκταση του παραδοσιακού Markdown: Παρέχει επιπλέον λειτουργίες πέρα από τις βασικές δυνατότητες του Markdown, όπως:

* Πλαίσια περιεχομένου (directives): Επιτρέπουν τη δημιουργία ειδικών στοιχείων, όπως προειδοποιήσεις, συμβουλές, κουτιά κειμένου, κ.λπ.
* Ρυθμίσεις κειμένου (roles): Ενσωματωμένες λειτουργίες για σύνθετα κείμενα ή μαθηματικά σύμβολα.
* Σύνθεση μαθηματικών: Υποστήριξη για LaTeX και MathJax για τη γραφή μαθηματικών εξισώσεων.

1. Προχωρημένα χαρακτηριστικά τεκμηρίωσης: Χρησιμοποιείται για τη συγγραφή τεχνικών εγγράφων και βιβλίων με δομή που περιλαμβάνει υποκεφάλαια, πίνακες, αναφορές, κώδικα, και διαγράμματα.
2. Διαλειτουργικότητα με reStructuredText και Sphinx: Το MyST Markdown είναι σχεδιασμένο ώστε να υποστηρίζει λειτουργίες που συναντάμε στο Sphinx, ένα εργαλείο που χρησιμοποιείται για τη δημιουργία τεκμηρίωσης σε Python projects. Αυτό σημαίνει ότι μπορείς να μετατρέπεις αρχεία .ipynb και .md σε HTML, PDF, ή άλλες μορφές χρησιμοποιώντας Sphinx.
3. Οι βιβλιοθήκες που χρησιμοποιήθηκαν στην εργασία παρέχουν εξαιρετικές δυνατότητες για την επεξεργασία δεδομένων, την προσομοίωση αλγορίθμων και την οπτικοποίηση αποτελεσμάτων. Εδώ θα αναλύσουμε περαιτέρω τις δυνατότητες αυτών των εργαλείων:

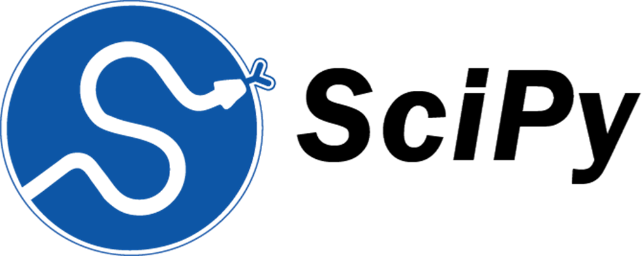
**NumPy**

Η NumPy είναι μια από τις πιο δημοφιλείς βιβλιοθήκες στην επιστημονική κοινότητα της Python. Παρέχει εργαλεία για αποδοτική διαχείριση μεγάλων πολυδιάστατων πινάκων (arrays) και συναρτήσεις που διευκολύνουν μαθηματικούς υπολογισμούς υψηλής απόδοσης. Έχει βελτιστοποιηθεί για ταχύτατες πράξεις, όπως ο πολλαπλασιασμός πινάκων και οι πράξεις σε διανύσματα, γεγονός που την καθιστά ιδανική για εφαρμογές που απαιτούν έντονη αριθμητική υπολογιστική ικανότητα, όπως οι αλγόριθμοι επεξεργασίας σημάτων που χρησιμοποιούνται στην εργασία.



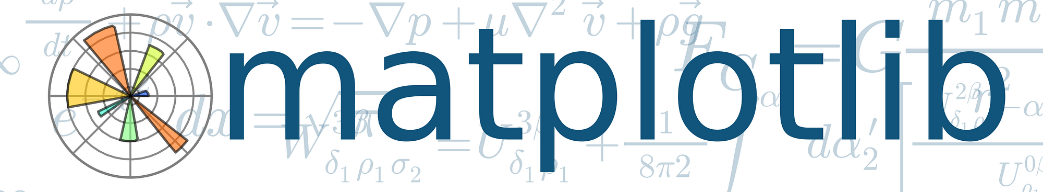
**SciPy**

Η SciPy επεκτείνει τη λειτουργικότητα της NumPy με πρόσθετες δυνατότητες για την ανάλυση επιστημονικών δεδομένων και την επίλυση προβλημάτων σε διάφορους τομείς, όπως γραμμική άλγεβρα, βελτιστοποίηση, ολοκληρώματα, και στατιστική. Στην εργασία, χρησιμοποιήθηκε κυρίως για την επίλυση γραμμικών συστημάτων εξισώσεων και για τον μετασχηματισμό Fourier, που αποτελεί κρίσιμο εργαλείο για την ανάλυση σημάτων στις τηλεπικοινωνίες.



**Matplotlib**

Η Matplotlib επιτρέπει τη δημιουργία διαγραμμάτων και γραφημάτων για την οπτικοποίηση των δεδομένων και των αποτελεσμάτων. Με τη χρήση της, μπορούμε να απεικονίσουμε πειραματικά δεδομένα, καμπύλες προσαρμογής και αποτελέσματα προσομοιώσεων με πολύ ευέλικτο και προσαρμοσμένο τρόπο. Αυτή η βιβλιοθήκη παρέχει μεγάλη ποικιλία τύπων γραφημάτων, από απλές γραμμές και διαγράμματα μέχρι τρισδιάστατες απεικονίσεις.

****

**IPython**

Το IPython είναι ένα διαδραστικό περιβάλλον ανάπτυξης κώδικα, το οποίο επιτρέπει την άμεση εκτέλεση εντολών Python. Μέσα από αυτό το περιβάλλον, οι χρήστες μπορούν να εκτελούν εντολές βήμα προς βήμα, να παρακολουθούν τα αποτελέσματα και να κάνουν γρήγορες διορθώσεις. Αυτό βοηθά σημαντικά στην ανάπτυξη αλγορίθμων και στην πειραματική διερεύνηση, καθώς προσφέρει μεγαλύτερη ευελιξία και ανατροφοδότηση από τα παραδοσιακά περιβάλλοντα ανάπτυξης.

Η βιβλιοθήκη Ipywidgets προσθέτει διαδραστικά στοιχεία στα Jupyter notebooks, όπως sliders, κουμπιά και πεδία κειμένου, τα οποία επιτρέπουν στους χρήστες να αλληλεπιδρούν με τους αλγορίθμους σε πραγματικό χρόνο. Αυτή η δυνατότητα διευκολύνει τη διαδικασία κατανόησης των μοντέλων και της συμπεριφοράς τους, δίνοντας τη δυνατότητα παραμετροποίησης των αλγορίθμων μέσω διαδραστικών widgets, χωρίς την ανάγκη να τροποποιείται ο κώδικας κάθε φορά.

**Requests**

Η βιβλιοθήκη Requests χρησιμοποιείται για την αποστολή αιτημάτων HTTP, επιτρέποντας την εύκολη και αποδοτική επικοινωνία με εξωτερικές διαδικτυακές υπηρεσίες και API. Αυτό είναι σημαντικό σε περιπτώσεις που χρειαζόμαστε δεδομένα από απομακρυσμένες πηγές ή θέλουμε να ενσωματώσουμε δεδομένα τρίτων στη μελέτη μας, όπως για παράδειγμα στην ανάλυση πραγματικών δεδομένων τηλεπικοινωνιών.

**Scikit-commpy**

Η Scikit-commpy είναι μια βιβλιοθήκη ειδικά σχεδιασμένη για την προσομοίωση και την ανάλυση συστημάτων επικοινωνίας. Περιέχει διάφορους αλγορίθμους για τη μετάδοση και λήψη σημάτων, όπως διαμορφώσεις (modulations), φίλτρα, καθώς και εργαλεία για την εκτίμηση του Bit Error Rate (BER). Η χρήση της σε αυτή την εργασία ήταν απαραίτητη για την προσομοίωση διαφόρων μεθόδων διαμόρφωσης σημάτων και για την αξιολόγηση της ποιότητας μετάδοσης, κάτι που αποτελεί βασικό κομμάτι της ανάλυσης.

**Thebe**

Το Thebe επιτρέπει την ενσωμάτωση κώδικα και διαδραστικών λειτουργιών σε ιστοσελίδες μέσω Jupyter kernels, δίνοντας τη δυνατότητα στους χρήστες να εκτελούν κώδικα απευθείας από τον browser τους. Αυτή η δυνατότητα καθιστά το περιβάλλον πιο διαδραστικό και προσφέρει τη δυνατότητα στους αναγνώστες της εργασίας να πειραματιστούν με τους αλγορίθμους και τα δεδομένα σε πραγματικό χρόνο, χωρίς να χρειάζεται να εγκαταστήσουν τα απαραίτητα εργαλεία στον υπολογιστή τους.

**SciPy Signal Processing (scipy.signal)**

Η βιβλιοθήκη scipy.signal είναι ένα ισχυρό εργαλείο για την επεξεργασία και ανάλυση σημάτων, παρέχοντας ένα σύνολο συναρτήσεων για την εφαρμογή φίλτρων, τη συνέλιξη, την αποδιαμόρφωση και την ανάλυση συχνοτήτων. Αποτελεί ένα από τα πιο χρήσιμα υποσύνολα της βιβλιοθήκης SciPy και χρησιμοποιείται ευρέως σε διάφορους τομείς της επιστήμης και της μηχανικής, όπως οι τηλεπικοινωνίες, η επεξεργασία εικόνας, οι βιοϊατρικές εφαρμογές και η ανάλυση συστημάτων ελέγχου.

Δυνατότητες

* *Φιλτράρισμα σημάτων*: Η scipy.signal παρέχει συναρτήσεις για το σχεδιασμό και την εφαρμογή ψηφιακών φίλτρων. Αυτά τα φίλτρα περιλαμβάνουν FIR (Finite Impulse Response) και IIR (Infinite Impulse Response) φίλτρα, τα οποία μπορούν να χρησιμοποιηθούν για την απομάκρυνση ανεπιθύμητων συχνοτήτων ή τη βελτίωση συγκεκριμένων στοιχείων του σήματος. Η συνάρτηση firwin, για παράδειγμα, χρησιμοποιείται για το σχεδιασμό φίλτρων FIR.
* *Συνέλιξη (Convolution):* Το εργαλείο αυτό είναι κρίσιμο για την ανάλυση και την επεξεργασία σημάτων. Η συνάρτηση convolve εφαρμόζει τη διαδικασία της συνέλιξης, η οποία είναι ιδιαίτερα σημαντική στη γραμμική ανάλυση συστημάτων και στη σχεδίαση φίλτρων.
* *Ανάλυση Συχνοτήτων:* Χρησιμοποιώντας μεθόδους όπως η συνάρτηση welch, η scipy.signal μπορεί να υπολογίσει την πυκνότητα φάσματος ισχύος (Power Spectral Density - PSD) ενός σήματος, η οποία είναι απαραίτητη για την ανάλυση των συχνοτήτων που υπάρχουν σε ένα σήμα. Αυτή η δυνατότητα είναι ιδιαίτερα χρήσιμη σε εφαρμογές όπως η φασματική ανάλυση σημάτων.
* *Ψηφιακή Διαμόρφωση και Αποδιαμόρφωση:* Η scipy.signal περιλαμβάνει εργαλεία για την επεξεργασία σημάτων με μεθόδους διαμόρφωσης και αποδιαμόρφωσης, τα οποία χρησιμοποιούνται συχνά σε τηλεπικοινωνιακές εφαρμογές.
* *Πολυ-ρυθμιστική Επεξεργασία Σήματος:* Με τη χρήση της συνάρτησης upfirdn, η βιβλιοθήκη επιτρέπει την εφαρμογή πολυ-ρυθμιστικής επεξεργασίας σημάτων (multirate signal processing). Αυτή η τεχνική χρησιμοποιείται για την αλλαγή της συχνότητας δειγματοληψίας, κάτι που είναι κρίσιμο σε συστήματα επεξεργασίας τηλεπικοινωνιακών σημάτων και ψηφιακών συστημάτων ήχου.
* *Σχεδίαση και Προσομοίωση Φίλτρων:* Οι συναρτήσεις όπως remez και firwin2 επιτρέπουν τον σχεδιασμό φίλτρων βέλτιστης μορφής, ενώ η συνάρτηση freqz παρέχει την δυνατότητα ανάλυσης της απόκρισης συχνότητας ενός φίλτρου. Οι μέθοδοι αυτές είναι ιδιαίτερα χρήσιμες για τον σχεδιασμό φίλτρων που απαιτούν αυστηρές προδιαγραφές σε εφαρμογές όπως η μετάδοση σημάτων.

**Math**

Η βιβλιοθήκη **math** της Python είναι μια από τις βασικότερες βιβλιοθήκες που παρέχουν εργαλεία για την εκτέλεση μαθηματικών υπολογισμών. Πρόκειται για μια εξαιρετικά αποδοτική και απαραίτητη βιβλιοθήκη σε επιστημονικούς και αριθμητικούς υπολογισμούς, καθώς περιλαμβάνει πλήθος βασικών αλλά και εξειδικευμένων μαθηματικών συναρτήσεων που συναντώνται συχνά σε διάφορους κλάδους της μηχανικής, της φυσικής και των υπολογιστικών επιστημών.

**IO**

Η βιβλιοθήκη io της Python παρέχει εργαλεία για την είσοδο και έξοδο δεδομένων (input/output) σε μορφή ροών (streams), προσφέροντας έναν ευέλικτο και ενοποιημένο τρόπο διαχείρισης δεδομένων από και προς αρχεία, μνήμη και άλλες πηγές. Οι ροές αυτές μπορούν να χρησιμοποιηθούν για να διαβάσουν ή να γράψουν δεδομένα σε μορφή κειμένου ή δυαδικών δεδομένων, ανάλογα με τις ανάγκες της εφαρμογής. Η βιβλιοθήκη io είναι ιδιαίτερα σημαντική για την διαχείριση αρχείων μεγάλου όγκου, την επικοινωνία δικτυακών εφαρμογών, καθώς και για την αποτελεσματική επεξεργασία δεδομένων σε συστήματα υψηλής απόδοσης.

**Time**

Η βιβλιοθήκη **time** της Python προσφέρει μια ποικιλία συναρτήσεων που σχετίζονται με την παρακολούθηση, μέτρηση και χειρισμό του χρόνου. Οι δυνατότητες της βιβλιοθήκης καλύπτουν πολλές ανάγκες, όπως η μέτρηση χρονικών διαστημάτων, η αναμονή (sleep) συγκεκριμένων χρονικών περιόδων και η μετατροπή χρονοσφραγίδων (timestamps) σε διάφορες μορφές. Η βιβλιοθήκη αυτή είναι ιδιαίτερα χρήσιμη σε εφαρμογές που απαιτούν συγχρονισμό, χρονόμετρα ή τον υπολογισμό της απόδοσης και της διάρκειας εκτέλεσης διεργασιών.

**Warnings**

Η βιβλιοθήκη **warnings** της Python παρέχει εργαλεία για την καταγραφή, διαχείριση και προσαρμογή των προειδοποιήσεων (warnings) που παράγονται κατά την εκτέλεση ενός προγράμματος. Σε αντίθεση με τα σφάλματα, οι προειδοποιήσεις δεν τερματίζουν το πρόγραμμα, αλλά ενημερώνουν τον προγραμματιστή για πιθανά ζητήματα, όπως η χρήση αποσυρμένων συναρτήσεων, ανεπιθύμητες τιμές παραμέτρων ή άλλες καταστάσεις που δεν είναι κρίσιμες για την εκτέλεση αλλά μπορεί να οδηγήσουν σε προβλήματα στο μέλλον.

Η warnings είναι χρήσιμη για τον εντοπισμό και τη διόρθωση πιθανών προβλημάτων στον κώδικα χωρίς να διακόπτεται η λειτουργία της εφαρμογής, καθώς και για την προσαρμογή της συμπεριφοράς της εφαρμογής όταν αντιμετωπίζει συγκεκριμένα προειδοποιητικά μηνύματα.

Με τη συνδυαστική χρήση αυτών των εργαλείων, η εργασία απέκτησε διαδραστικό χαρακτήρα, καθιστώντας τα δεδομένα και τις αναλύσεις προσιτές και εύκολα κατανοητές, ενώ παράλληλα παρέχει τη δυνατότητα στους αναγνώστες να επαναλάβουν πειράματα και να τροποποιήσουν παραμέτρους, κάτι που ενισχύει την εκπαιδευτική της αξία.

Κεφάλαιο 3

# Μεθοδολογία, Σχεδίαση και Υλοποίηση διαδραστικών στοιχείων

## 3.1 Εισαγωγή

Σε αυτό το κεφάλαιο, αρχικά, παρουσιάζεται η συνολική μεθοδολογία που ακολουθήθηκε για την ολοκληρωμένη ανάπτυξη της διαδικτυακής διαδραστικής εφαρμογής. Στη συνέχεια, παρατίθενται τα τμήματα κώδικα που ενσωματώνουν διαδραστικές λειτουργίες, οι οποίες συνιστούν τη βάση για την ολοκλήρωση του έργου.

## 3.2 Μέθοδος υλοποίησης του συνολικού project

Η παρούσα εργασία περιγράφει αναλυτικά τα πρώτα στάδια υλοποίησης της διαδικτυακής εφαρμογής, η οποία αναπτύχθηκε σε συνεργασία με τον Aραβαντινού Καρλάτου Σωτήρη. Ειδικότερα, στο παρόν τεύχος παρατίθεται η ανάλυση των βημάτων 10 - 18, τα οποία αποτελούν η συνέχεια και η ολοκλήρωση της συνολικής εργασίας. Για περαιτέρω πληροφορίες σχετικά με τα αρχικά βήματα και το στήσιμο του έργου, ο αναγνώστης παραπέμπεται στην αντίστοιχη εργασία του Aραβαντινού Καρλάτου Σωτήρη.

1. Αναζήτηση, δοκιμή και τελική επιλογή των κατάλληλων εργαλείων για την ολοκλήρωση του project.
2. Δημιουργία αποθετηρίου στο GitHub για τη διαχείριση των αρχείων του έργου.
3. Κατασκευή βασικών κομματιών κώδικα python γύρω από την θεωρία και τις εργαστηριακές ασκήσεις του μαθήματος "Ψηφιακές Επικοινωνίες Ι", με διαδραστικές δυνατότητες. (Βάση μας ο κώδικας matlab του μαθήματος)
4. Ανάπτυξη και διαμόρφωση του Jupyter Book ως εργαλείο παρουσίασης και εκτέλεσης του εκπαιδευτικού υλικού.
5. Τροποποίηση του αρχείου config.yml για την ενσωμάτωση της επιλογής binder, επιτρέποντας την εκτέλεση κώδικα σε πραγματικό χρόνο μέσω ενός εξωτερικού kernel.
6. Δημιουργία του αρχείου requirements.txt, περιλαμβάνοντας όλες τις απαιτούμενες βιβλιοθήκες Python για την εξασφάλιση της λειτουργικότητας του kernel μέσω του MyBinder.
7. Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για τη δημιουργία των αρχείων της ιστοσελίδας.
8. Χρήση του GitHub Pages για την ανάπτυξη και φιλοξενία της διαδικτυακής εφαρμογής.
9. Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για την δημοσίευση της ιστοσελίδας στο GitHub Page που δημιουργήσαμε.
10. **Αναδιάρθρωση της δομής του Jupyter Book μέσω αλλαγών στο αρχείο toc.yml, ώστε να ευθυγραμμίζεται με τη διάρθρωση των εργαστηριακών ασκήσεων του μαθήματος "Ψηφιακές Επικοινωνίες Ι".**
11. **Μετάφραση του περιεχομένου των εργαστηριακών ασκήσεων στα Αγγλικά.**
12. **Ενσωμάτωση του μεταφρασμένου περιεχομένου στα Jupyter Notebooks που δημιουργήθηκαν. Ένα για κάθε άσκηση.**
13. **Εντοπισμός των σημείων που απαιτούν διαδραστικότητα και ενσωμάτωση από τον υπάρχον κώδικα, ή και εκ νέου σχεδιασμός, αντίστοιχων λειτουργιών.**
14. **Μετατροπή του υπάρχοντος βοηθητικού κώδικα από MATLAB σε Python, με χρήση των βιβλιοθηκών ψηφιακής επικοινωνίας και γραφικής απεικόνισης δεδομένων. (Με την βοήθεια των κομματιών κώδικα από το βήμα 3)**
15. **Προσθήκη διαδραστικών λειτουργιών στον κώδικα Python μέσω των κατάλληλων βιβλιοθηκών. (Με την βοήθεια των κομματιών κώδικα από το βήμα 3)**
16. **Ανανέωση του αρχείου requirements.txt, ώστε να περιλαμβάνει τις τελικές βιβλιοθήκες python που χρησιμοποιήθηκαν.**
17. **Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για τη δημιουργία των αρχείων της ιστοσελίδας.**
18. **Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για την δημοσίευση της ιστοσελίδας στο GitHub Page που δημιουργήσαμε.**

## 3.3 Τα πρώτα βήματα για το στήσιμο της σελίδας

Η αναδιάρθρωση της δομής του Jupyter Book έγινε προκειμένου να ευθυγραμμιστεί με τις διατάξεις των εργαστηριακών ασκήσεων του μαθήματος "Ψηφιακές Επικοινωνίες Ι". Η δομή αυτή είναι ουσιώδης για την οργάνωση της διδακτικής ύλης και την εύκολη πρόσβαση στις ασκήσεις, επιτρέποντας στους φοιτητές να ακολουθήσουν μια λογική ροή μάθησης. Στο πλαίσιο της αναβάθμισης της εκπαιδευτικής διαδικασίας, προχωρήσαμε στη μετάφραση του περιεχομένου των εργαστηριακών ασκήσεων από τα ελληνικά στα αγγλικά, με στόχο τη διευκόλυνση της πρόσβασης διεθνών φοιτητών και ερευνητών. Το μεταφρασμένο υλικό έχει ενσωματωθεί στα Jupyter Notebooks που αναπτύχθηκαν για κάθε άσκηση

Η μετατροπή των κωδικών από MATLAB σε Python βασίστηκε στη χρήση βιβλιοθηκών για την ψηφιακή επεξεργασία σήματος και την οπτικοποίηση δεδομένων. Η αλλαγή αυτή επέφερε ευελιξία και ευκολότερη πρόσβαση στις λειτουργίες, καθώς η Python είναι πιο δημοφιλής και προσφέρει καλύτερες διαδραστικές δυνατότητες μέσω εργαλείων όπως το Matplotlib.

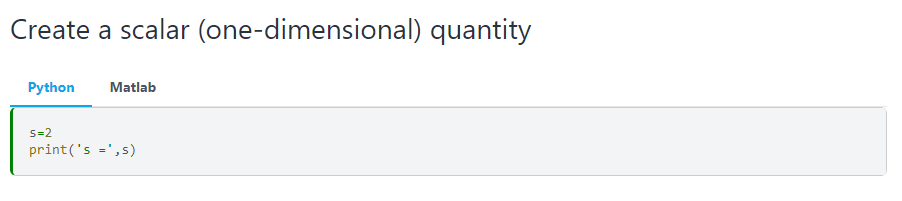
## 3.4 Διαδραστικά στοιχεία που χρησιμοποιήθηκαν

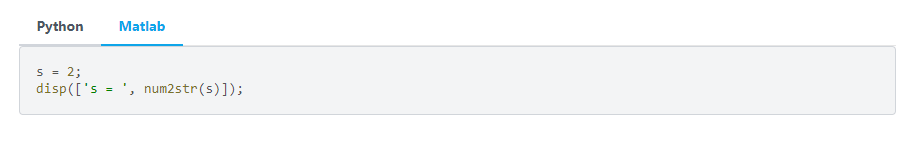
Παράλληλα, κατά την ενσωμάτωση του περιεχομένου, εντοπίσαμε σημεία στα οποία η διαδραστικότητα κρίνεται απαραίτητη, ώστε να βελτιωθεί η κατανόηση των εννοιών και η ενεργός συμμετοχή των χρηστών. Τα σημεία αυτά αποτελούν σημαντικούς κόμβους για την περαιτέρω ανάπτυξη των δυνατοτήτων των notebooks, με σκοπό την ενσωμάτωση διαδραστικών στοιχείων, όπως widgets, διαδραστικές γραφικές παραστάσεις και δυναμικά διαγράμματα. Αυτή η διαδικασία στοχεύει στη δημιουργία ενός πιο ελκυστικού και αποτελεσματικού μαθησιακού περιβάλλοντος.

### 3.4.1 Καρτέλες (Tabs)

Στη σύγχρονη εκπαίδευση και τεχνολογική τεκμηρίωση, η χρήση διαδραστικών εργαλείων αποτελεί κεντρικό άξονα για τη διευκόλυνση της κατανόησης και την ενίσχυση της αφομοίωσης της γνώσης. Ένα από τα πιο αποτελεσματικά διαδραστικά στοιχεία είναι η δομή των καρτελών (tabs), η οποία προσφέρει ευελιξία και δυνατότητα δυναμικής παρουσίασης περιεχομένου. Οι καρτέλες επιτρέπουν τη διαχείριση μεγάλου όγκου δεδομένων και πληροφορίας μέσα σε περιορισμένο χώρο, χωρίς να επιβαρύνεται η εμπειρία του χρήστη με την πλοήγηση ανάμεσα σε διαφορετικές σελίδες ή ενότητες.

Η διαδραστικότητα των καρτελών έγκειται κυρίως στην δυνατότητά τους να προσφέρουν ταχύτατη εναλλαγή μεταξύ διαφόρων τμημάτων περιεχομένου. Αυτή η εναλλαγή διευκολύνει την παράλληλη μελέτη και σύγκριση διαφορετικών προσεγγίσεων, π.χ. μεθοδολογιών, αλγορίθμων ή γλωσσών προγραμματισμού, κάνοντας την πληροφορία άμεσα διαθέσιμη χωρίς να απαιτείται αναζήτηση σε διαφορετικά έγγραφα ή πλατφόρμες. Η ενσωμάτωση καρτελών σε μια εκπαιδευτική ή τεχνική πλατφόρμα μπορεί να αυξήσει δραστικά την αλληλεπίδραση του χρήστη με το περιεχόμενο, καθώς δίνει τη δυνατότητα να διαλέγει και να εξετάζει μόνο το τμήμα που τον ενδιαφέρει εκείνη τη στιγμή, παραμερίζοντας άσχετες πληροφορίες.

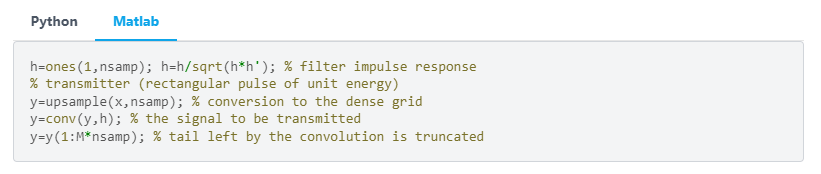
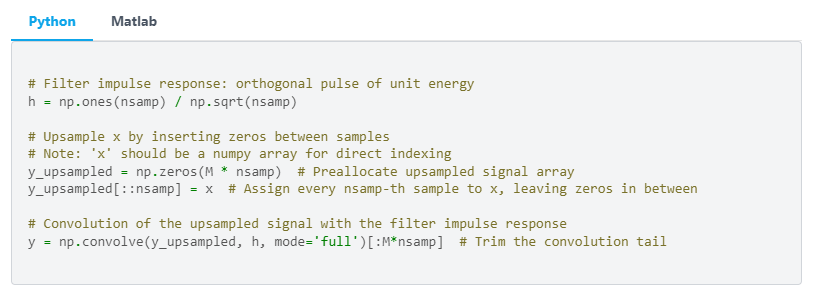




Σχήμα 12: 1ο Παράδειγμα Καρτελών (Tabs)

Σε ένα επιστημονικό πλαίσιο, η χρήση καρτελών επιτρέπει την προβολή διαφορετικών σεναρίων ή υλοποιήσεων του ίδιου προβλήματος, με την ίδια βάση δεδομένων ή αλγορίθμου, σε διαφορετικές καρτέλες. Αυτή η δυνατότητα είναι ιδιαίτερα χρήσιμη όταν εξετάζονται αλληλένδετες προσεγγίσεις, επιτρέποντας στον αναγνώστη να παρακολουθήσει την πορεία της ανάλυσης με διακριτό αλλά παράλληλο τρόπο. Για παράδειγμα, σε μια επιστημονική δημοσίευση σχετική με την επεξεργασία ψηφιακών σημάτων, η χρήση καρτελών μπορεί να επιτρέψει την προβολή της μαθηματικής ανάλυσης ενός φίλτρου σε μία καρτέλα, ενώ μια άλλη καρτέλα μπορεί να παρουσιάζει τον αλγόριθμο σε κώδικα Python, και μια τρίτη να συγκρίνει την υλοποίηση του ίδιου φίλτρου σε MATLAB.

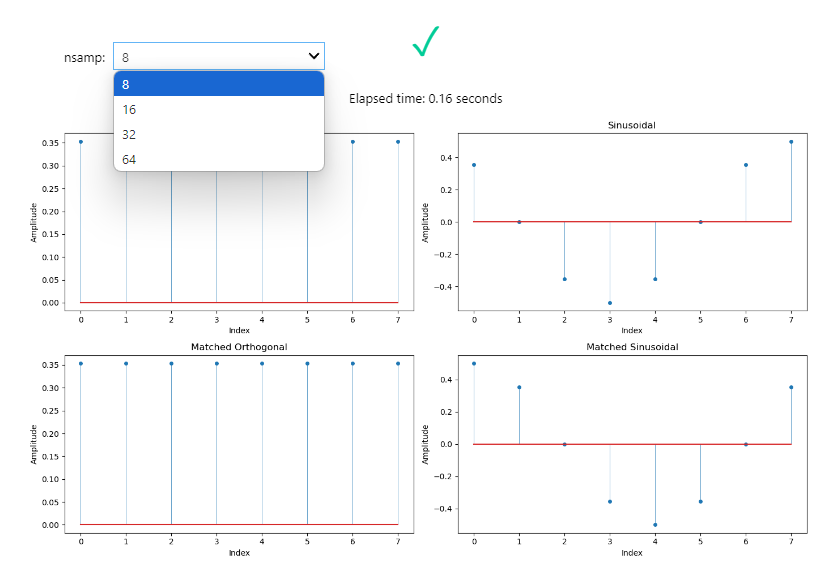
Η χρήση καρτελών ενισχύει τη διαδραστικότητα και προωθεί την εμπλοκή του χρήστη με το περιεχόμενο, καθώς μπορεί να επιλέξει το τμήμα που του ταιριάζει περισσότερο. Σε τεχνικά έγγραφα, αυτό το είδος οργάνωσης παρέχει μια καθαρή και λειτουργική δομή που μπορεί να χρησιμοποιηθεί για τη διεπιστημονική μάθηση ή την εκπαιδευτική ανάλυση διαφορετικών τεχνολογιών, ενθαρρύνοντας την αναλυτική σκέψη και την κριτική σύγκριση των δεδομένων.



Σχήμα : 2ο Παράδειγμα Καρτελών (Tabs)'

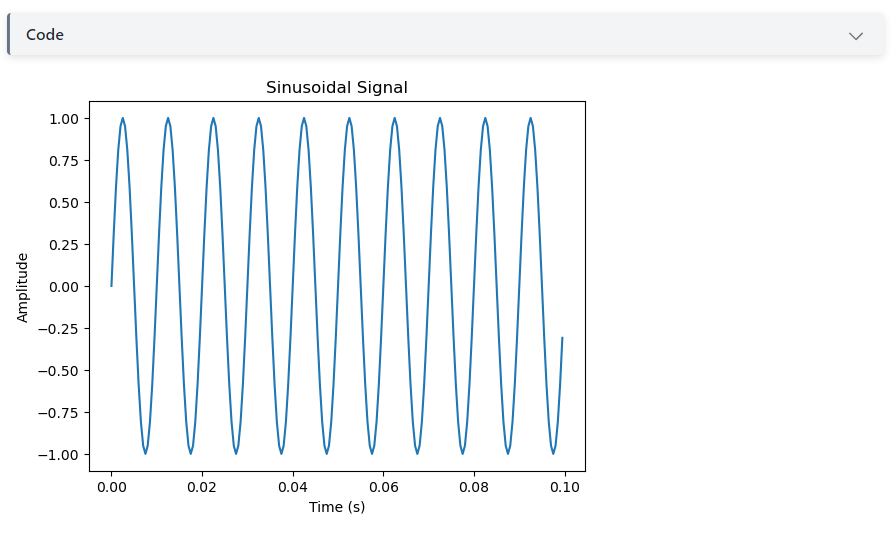
### 3.4.2 Αναπτυσσόμενα μενού (Dropdown)

Η χρήση διαδραστικών στοιχείων σε τεχνικά και επιστημονικά κείμενα αποτελεί μια στρατηγική που ενισχύει την εμπειρία του αναγνώστη, διευκολύνοντας την κατανόηση και την οργάνωση μεγάλου όγκου πληροφοριών. Ένα από τα σημαντικότερα εργαλεία που χρησιμοποιείται για αυτόν τον σκοπό είναι το dropdown, το οποίο επιτρέπει την απόκρυψη και εμφάνιση πληροφοριών με τρόπο που διατηρεί τη δομή του κειμένου καθαρή και ευανάγνωστη. Συγκεκριμένα, το dropdown επιτρέπει την προσωρινή απόκρυψη μεγάλων κομματιών κώδικα, περιγραφών ή τεχνικών αναλύσεων, και την εμφάνισή τους μόνο όταν ο αναγνώστης επιλέξει να τα επεκτείνει. Αυτή η δυνατότητα συμβάλλει στη μείωση της οπτικής επιβάρυνσης του κειμένου και επιτρέπει την παρουσίαση πληροφοριών σε ένα οργανωμένο και δομημένο πλαίσιο.



Σχήμα 14: 1ο Παράδειγμα Αναπτυσσόμενου μενού (dropdown)

Το dropdown εργαλείο είναι ιδιαίτερα αποτελεσματικό όταν το κείμενο περιέχει μεγάλα μπλοκ κώδικα ή εκτενείς επεξηγηματικές παραγράφους που, αν εμφανίζονταν πλήρως εξ αρχής, θα μπορούσαν να δυσχεράνουν την ανάγνωση του υπόλοιπου περιεχομένου. Η ευελιξία που παρέχει είναι ιδανική για τεχνικά έγγραφα και εκπαιδευτικά υλικά, όπου η παρουσία μεγάλου όγκου δεδομένων είναι συχνή. Οι αναγνώστες έχουν την ευχέρεια να επικεντρώνονται μόνο στα τμήματα που τους ενδιαφέρουν άμεσα, ενώ η δυνατότητα να επεκτείνουν το dropdown όποτε το επιθυμούν προσφέρει μια αίσθηση ελέγχου και διάδρασης με το κείμενο.



Σχήμα : 2ο Παράδειγμα Αναπτυσσόμενου μενού (dropdown)

**Ο συνδυασμός του Dropdown με τις Καρτέλες (Tabs)**

Η ενσωμάτωση του dropdown με τις καρτέλες (tabs) αποτελεί έναν ακόμη πιο προηγμένο τρόπο παρουσίασης περιεχομένου, ιδανικό για περιπτώσεις όπου χρειάζεται να συγκριθούν διαφορετικές προσεγγίσεις, τεχνολογίες ή γλώσσες προγραμματισμού σε ένα ενιαίο περιβάλλον. Οι καρτέλες επιτρέπουν τη διαχωρισμένη παρουσίαση περιεχομένου, όπως διαφορετικές γλώσσες κώδικα, ενώ το dropdown συμπυκνώνει τον συνολικό όγκο του κειμένου, δίνοντας την επιλογή στον αναγνώστη να επεκτείνει μόνο το μέρος που τον ενδιαφέρει.

Για παράδειγμα, σε ένα τεχνικό κείμενο που παρουσιάζει την υλοποίηση ενός αλγορίθμου τόσο σε Python όσο και σε MATLAB, η χρήση καρτελών εντός του dropdown προσφέρει τη δυνατότητα επιλογής μεταξύ των δύο γλωσσών χωρίς να εμφανίζεται ολόκληρος ο κώδικας εξ αρχής. Αυτή η οργάνωση βοηθά στην αποφυγή υπερφόρτωσης πληροφοριών, προσφέροντας παράλληλα ευελιξία στην ανάγνωση. Οι καρτέλες εντός του dropdown διευκολύνουν τη γρήγορη εναλλαγή μεταξύ των γλωσσών και των υλοποιήσεων, προσφέροντας μια διαδραστική εμπειρία που προωθεί τη διεπιστημονική μάθηση και τη σύγκριση διαφορετικών προσεγγίσεων.

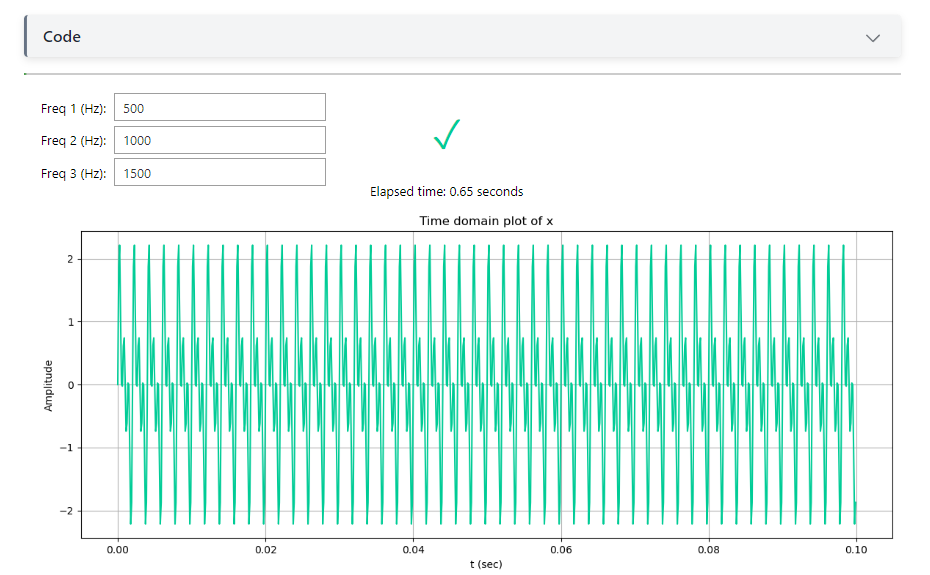
Συνολικά, ο συνδυασμός dropdown και tabs δημιουργεί ένα εξαιρετικά ευέλικτο, πολυδιάστατο εργαλείο που ενισχύει τη διάδραση, την οργάνωση και την αποτελεσματικότητα στην παρουσίαση τεχνικού υλικού, καθιστώντας το κατάλληλο τόσο για εκπαιδευτικά όσο και για επιστημονικά έγγραφα.

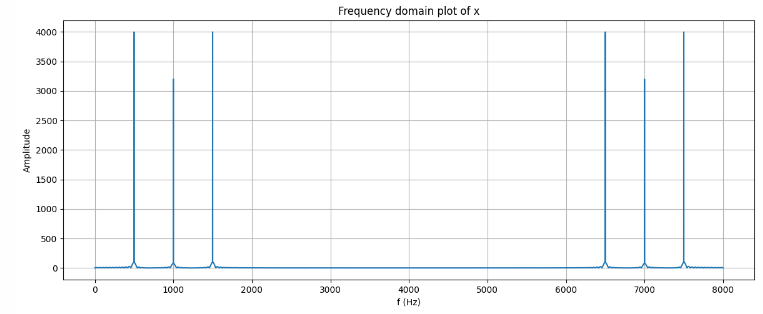
Σχήμα 16: Παράδειγμα συνδυασμού καρτελών και αναπτυσσόμενου μενού

### 3.4.3 Πεδίο κειμένου (Text box)

Η χρήση διαδραστικών στοιχείων σε ένα επιστημονικό ή εκπαιδευτικό περιβάλλον αποτελεί ένα από τα πιο σημαντικά εργαλεία για την κατανόηση σύνθετων εννοιών, ιδίως όταν πρόκειται για αναλυτικά μαθήματα που εμπλέκουν μαθηματική επεξεργασία και ανάλυση σημάτων. Τα text boxes, που λειτουργούν ως πεδία εισαγωγής τιμών, αποτελούν ένα ευέλικτο και ισχυρό εργαλείο για τη δημιουργία διαδραστικών εφαρμογών, προσφέροντας στους χρήστες τη δυνατότητα να εισάγουν τιμές και να βλέπουν άμεσα τις αλλαγές στα αποτελέσματα.

Σε ένα εργαστηριακό περιβάλλον όπως αυτό του μαθήματος Ψηφιακών Επικοινωνιών Ι, τα text boxes επιτρέπουν την εισαγωγή μεταβλητών όπως συχνότητες σημάτων ή τιμές συχνότητας δειγματοληψίας, προσφέροντας μια προσαρμοσμένη εμπειρία για κάθε χρήστη. Κάθε αλλαγή στις τιμές που εισάγονται μέσω των text boxes ανανεώνει δυναμικά την αναπαράσταση του σήματος και τα σχετικά γραφήματα, χωρίς να απαιτείται η χειροκίνητη επεξεργασία του κώδικα ή η επανεκκίνηση της διαδικασίας. Η διαδραστικότητα αυτή επιτρέπει στους φοιτητές και τους ερευνητές να εξετάσουν γρήγορα και αποτελεσματικά διαφορετικά σενάρια και παραμέτρους, διευκολύνοντας τη βαθύτερη κατανόηση των εννοιών που σχετίζονται με την ψηφιακή επεξεργασία σημάτων.

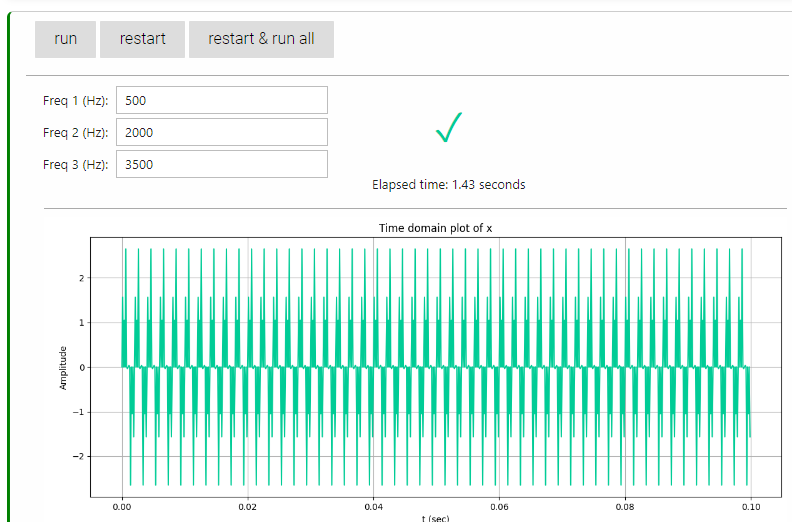


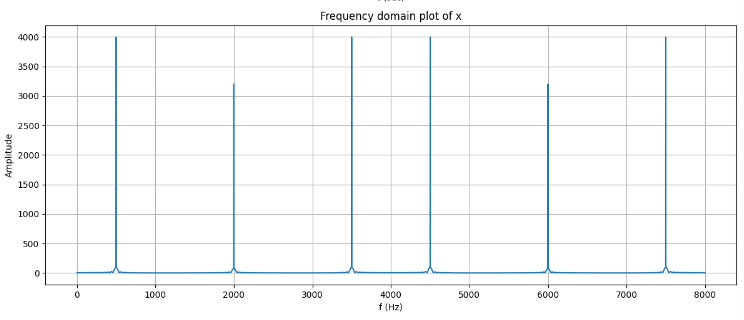


Σχήμα 17: 1ο Παράδειγμα πεδίου κειμένου

**Η Δυναμική Ενημέρωση με Text Boxes**

Η δυναμική ενημέρωση μέσω text boxes βασίζεται στην αλληλεπίδραση του χρήστη με την εφαρμογή, δίνοντας τη δυνατότητα άμεσης τροποποίησης των τιμών των μεταβλητών και επανασχεδιασμού των γραφημάτων σε πραγματικό χρόνο. Αυτό επιτυγχάνεται με την ενσωμάτωση κώδικα που συνδέει τα text boxes με τις αντίστοιχες παραμέτρους του σήματος, όπως οι συχνότητες. Σε ένα παράδειγμα όπου η ανάλυση αφορά ένα ημιτονοειδές σήμα, οι φοιτητές μπορούν να εισάγουν τιμές συχνοτήτων μέσω των text boxes, τα οποία είναι κωδικοποιημένα με ονόματα όπως freq1, freq2 και freq3 για την αντιστοίχιση με τις συχνότητες του σήματος. Αυτή η αλληλεπίδραση επιτρέπει την άμεση σύγκριση των αποτελεσμάτων με διαφορετικές τιμές και την παρακολούθηση του τρόπου που επηρεάζεται το σήμα σε πραγματικό χρόνο.

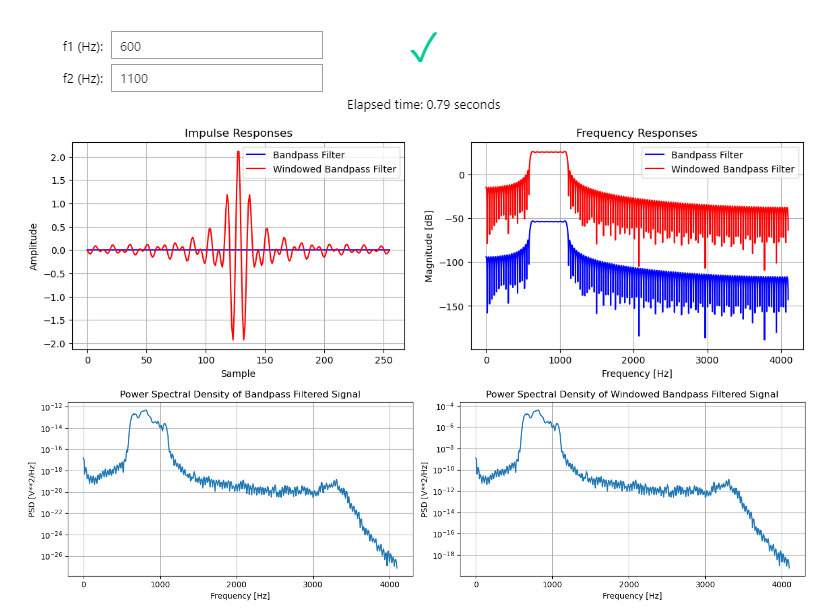




Σχήμα : 2ο Παράδειγμα πεδίου κειμένου

**Επιστημονική Τεκμηρίωση και Εκπαιδευτική Αξία**

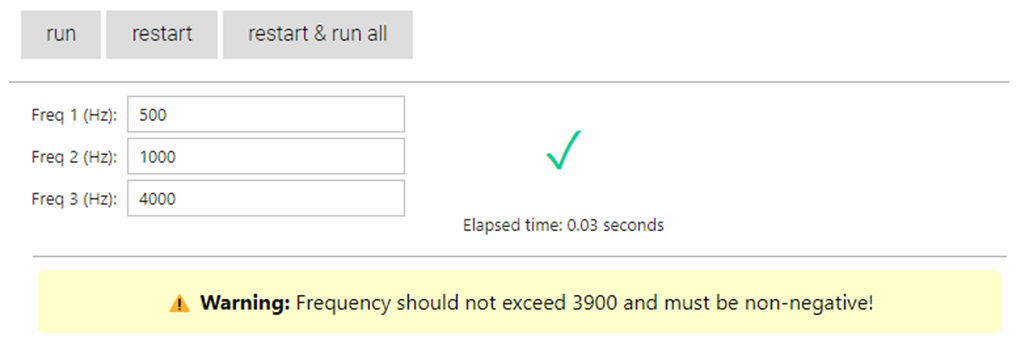
Η ενσωμάτωση text boxes σε εργαστηριακές εφαρμογές προσφέρει ουσιαστικά πλεονεκτήματα σε επιστημονικές και εκπαιδευτικές δραστηριότητες. Από επιστημονική άποψη, η δυνατότητα άμεσης ενημέρωσης των τιμών των παραμέτρων χωρίς την ανάγκη επανεκκίνησης του κώδικα βελτιώνει την αποτελεσματικότητα της ανάλυσης και τη σύγκριση των αποτελεσμάτων. Οι χρήστες μπορούν να δοκιμάσουν πολλές διαφορετικές παραμέτρους γρήγορα, ενθαρρύνοντας την ανάλυση σε βάθος. Στην εκπαιδευτική διαδικασία, αυτό το εργαλείο προσφέρει ένα πιο ευέλικτο και φιλικό προς το χρήστη περιβάλλον μάθησης, όπου οι φοιτητές μπορούν να πειραματιστούν και να κατανοήσουν καλύτερα τις επιπτώσεις των αλλαγών στις παραμέτρους των σημάτων.



Σχήμα 19: 3ο Παράδειγμα πεδίου κειμένου

Συνολικά, η χρήση των text boxes ενισχύει τη διαδραστικότητα και την ευελιξία στην εκπαιδευτική και επιστημονική ανάλυση, παρέχοντας ένα εργαλείο που επιτρέπει την άμεση αλληλεπίδραση του χρήστη με το περιεχόμενο και τη δημιουργία δυναμικών γραφημάτων με βάση τις προτιμήσεις και τις ανάγκες του χρήστη.

Ο κώδικας ελέγχει αν οι τιμές των συχνοτήτων υπερβαίνουν το επιτρεπτό όριο (3900 Hz) ή αν είναι αρνητικές. Σε περίπτωση μη έγκυρης τιμής, εμφανίζεται προειδοποιητικό μήνυμα και η οπτικοποίηση διακόπτεται, προστατεύοντας το σύστημα από σφάλματα ή μη ρεαλιστικές εισόδους.



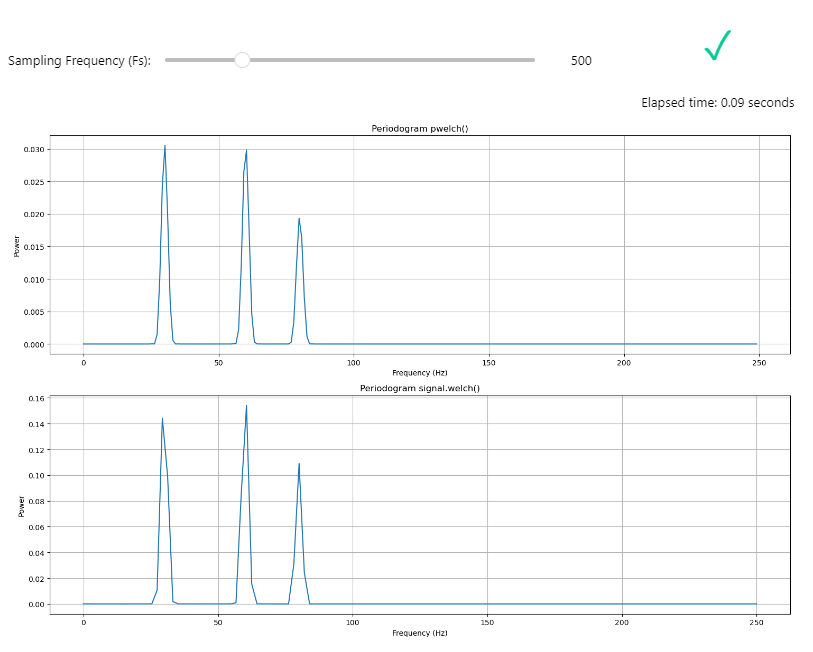
Σχήμα 20: Παράδειγμα ελέγχου πεδίου κειμένου

### 3.4.4 Ολισθητής (Slider)

Οι sliders (ολισθητές) αποτελούν ένα από τα πιο διαδραστικά και εύχρηστα εργαλεία σε εκπαιδευτικά και επιστημονικά περιβάλλοντα, καθώς επιτρέπουν στους χρήστες να προσαρμόζουν δυναμικά τις τιμές των παραμέτρων και να παρατηρούν άμεσα τις αλλαγές στα αποτελέσματα. Η χρήση των sliders προσφέρει έναν ευέλικτο και διαισθητικό τρόπο χειρισμού σύνθετων παραμέτρων, όπως η συχνότητα, το πλάτος ή η φάση των σημάτων, σε πραγματικό χρόνο, χωρίς να απαιτείται η άμεση εισαγωγή αριθμητικών τιμών ή η επαναφορά του κώδικα.

**Διαδραστικότητα μέσω Sliders**

Η διαδραστικότητα που προσφέρουν οι sliders ενισχύει σημαντικά την εμπειρία του χρήστη, καθώς επιτρέπει την προσαρμογή τιμών με τρόπο συνεχή και άμεσο. Η δυνατότητα να αλλάζουν οι τιμές των παραμέτρων σε πραγματικό χρόνο και να ενημερώνεται αυτόματα το αντίστοιχο διάγραμμα ή το αποτέλεσμα βοηθά τους χρήστες να αποκτήσουν μια διαισθητική κατανόηση των σχέσεων μεταξύ των παραμέτρων. Στα μαθήματα ψηφιακών επικοινωνιών και επεξεργασίας σημάτων, για παράδειγμα, οι sliders μπορούν να χρησιμοποιηθούν για τη ρύθμιση παραμέτρων όπως η συχνότητα δειγματοληψίας, το πλάτος ή η συχνότητα ενός σήματος, επιτρέποντας τη δυναμική αναπαράσταση του σήματος και των σχετικών γραφημάτων.

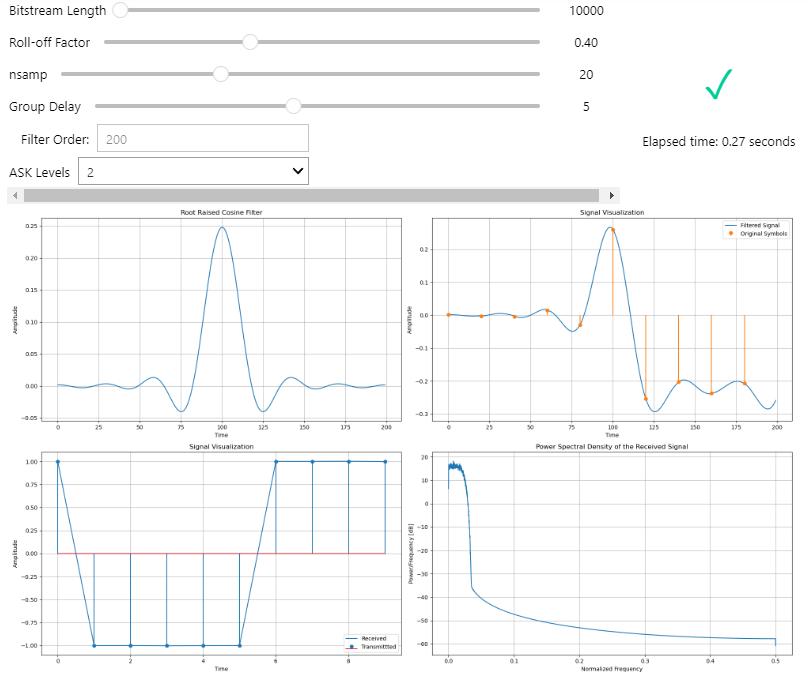


Σχήμα 21: 1ο Παράδειγμα ολισθητή

Οι sliders διασφαλίζουν ότι ο χρήστης μπορεί να δοκιμάσει πολλές διαφορετικές παραμέτρους γρήγορα και αποτελεσματικά, χωρίς την ανάγκη για επαναλαμβανόμενη χειροκίνητη εισαγωγή δεδομένων ή ανανέωση του κώδικα. Αυτό αυξάνει την παραγωγικότητα και την αμεσότητα στην ανάλυση των αποτελεσμάτων, ενώ παράλληλα ενθαρρύνει τον πειραματισμό και τη σύγκριση διαφορετικών σεναρίων.

**Δυναμική Ενημέρωση και Οπτικοποίηση**

Σε επιστημονικά πλαίσια, οι sliders μπορούν να χρησιμοποιηθούν για τη δυναμική ενημέρωση και οπτικοποίηση των δεδομένων. Η συνεχής μεταβολή των τιμών μιας μεταβλητής επιτρέπει την παρατήρηση της επίδρασης αυτής της μεταβολής στα αποτελέσματα σε πραγματικό χρόνο, γεγονός που βοηθά στην κατανόηση των μαθηματικών σχέσεων και των λειτουργιών των σημάτων. Για παράδειγμα, σε ένα εργαστηριακό περιβάλλον όπου αναλύεται ένα ημιτονοειδές σήμα, οι sliders επιτρέπουν τη συνεχή αλλαγή της συχνότητας και την άμεση οπτικοποίηση της επίδρασης αυτής στο σήμα.



Σχήμα 22: 2ο Παράδειγμα ολισθητή

Οι sliders που αντιστοιχούν σε συγκεκριμένες παραμέτρους, όπως η συχνότητα ή το πλάτος, επιτρέπουν την άμεση παρακολούθηση του τρόπου που οι μεταβολές αυτών των παραμέτρων αλλάζουν το σήμα ή τα γραφήματα που το συνοδεύουν. Αυτή η προσέγγιση καθιστά τον χρήστη πιο ενεργό στην ανάλυση, διευκολύνοντας τη σύγκριση και την κατανόηση σύνθετων σχέσεων.

**Επιστημονική και Εκπαιδευτική Αξία**

Η ενσωμάτωση sliders σε εκπαιδευτικά και επιστημονικά εργαλεία ενισχύει την εμπειρία μάθησης και ανάλυσης, παρέχοντας έναν πρακτικό και διαδραστικό τρόπο πειραματισμού με παραμέτρους. Στην επιστημονική τεκμηρίωση, οι sliders μπορούν να χρησιμοποιηθούν για την παρουσίαση ευέλικτων παραδειγμάτων, όπου ο αναγνώστης μπορεί να ρυθμίσει τις παραμέτρους για να κατανοήσει καλύτερα τις μεταβλητές και τα δεδομένα. Σε εκπαιδευτικό επίπεδο, οι sliders προσφέρουν μια ενεργή μορφή μάθησης, όπου οι φοιτητές μπορούν να εξετάσουν τις επιπτώσεις αλλαγών σε παραμέτρους χωρίς να απαιτείται η κατανόηση όλων των λεπτομερειών του κώδικα.

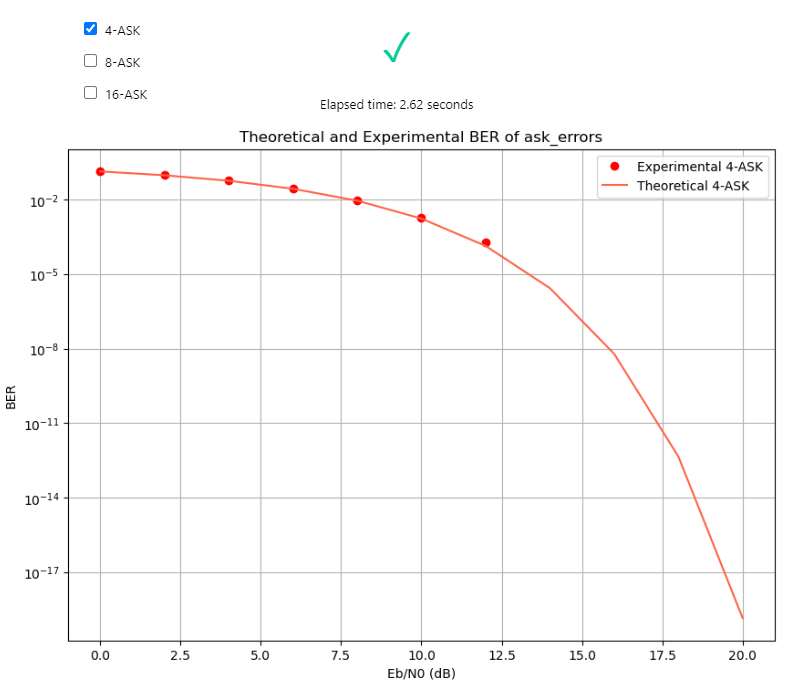
Συνολικά, οι sliders αποτελούν ένα ισχυρό εργαλείο για την αλληλεπίδραση του χρήστη με το περιεχόμενο, προσφέροντας δυνατότητες για πιο αποτελεσματική ανάλυση και κατανόηση σύνθετων επιστημονικών εννοιών και σχέσεων.

### 3.4.5 Πλαίσιο επιλογής (Checkbox)

Τα checkboxes αποτελούν ένα εξαιρετικά χρήσιμο και διαδραστικό εργαλείο σε εκπαιδευτικά και επιστημονικά περιβάλλοντα, ειδικά σε περιπτώσεις όπου ο χρήστης καλείται να επιλέξει μεταξύ πολλαπλών επιλογών ή να ενεργοποιήσει/απενεργοποιήσει συγκεκριμένες λειτουργίες. Το πλεονέκτημα των checkboxes έγκειται στην απλότητά τους, καθώς παρέχουν έναν άμεσο και οπτικά κατανοητό τρόπο για την ενεργοποίηση ή την απόρριψη μιας επιλογής χωρίς την ανάγκη εισαγωγής κειμένου ή πιο περίπλοκων αλληλεπιδράσεων.

**Διαδραστικότητα μέσω Checkboxes**

Η διαδραστικότητα των checkboxes βασίζεται στη δυνατότητα που παρέχουν στον χρήστη να επιλέξει ποια χαρακτηριστικά ή δεδομένα επιθυμεί να ενεργοποιήσει. Για παράδειγμα, σε ένα περιβάλλον ανάλυσης σημάτων, τα checkboxes μπορούν να χρησιμοποιηθούν για την επιλογή συγκεκριμένων φίλτρων ή γραφημάτων που θέλει να ενεργοποιήσει ο χρήστης. Κάθε επιλογή που γίνεται μέσω ενός checkbox ενημερώνει άμεσα το περιεχόμενο που παρουσιάζεται ή επεξεργάζεται στην εφαρμογή, χωρίς να απαιτείται η επανεκκίνηση της διαδικασίας.

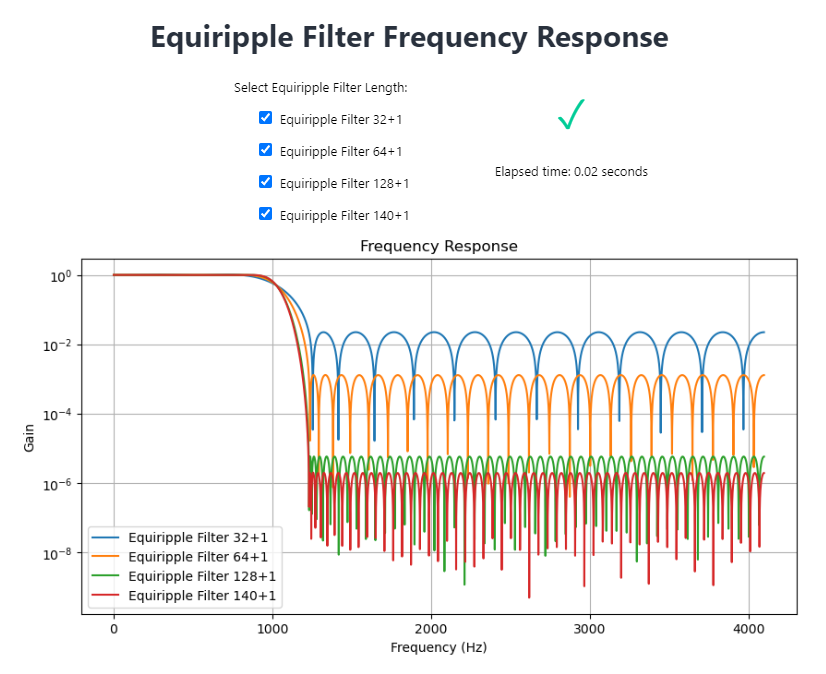


Σχήμα 23: 1ο Παράδειγμα πλαισίου επιλογής

Σε πρακτικό επίπεδο, αυτό σημαίνει ότι ο χρήστης μπορεί να επιλέξει να εμφανίσει ή να αποκρύψει συγκεκριμένα γραφήματα, να ενεργοποιήσει πρόσθετες λειτουργίες ή να περιλάβει διαφορετικές παραμέτρους στην ανάλυσή του. Αυτή η διαδραστικότητα επιτρέπει μια πιο ευέλικτη και προσαρμόσιμη εμπειρία, καθώς το περιεχόμενο προσαρμόζεται στις ανάγκες και τις προτιμήσεις του χρήστη.

**Εφαρμογές Checkboxes σε Επιστημονικά Πλαίσια**

Στην επιστημονική τεκμηρίωση, τα checkboxes χρησιμοποιούνται συχνά για να διευκολύνουν τη διαχείριση σύνθετων σεναρίων και δεδομένων. Ένα παράδειγμα αυτής της χρήσης είναι η παρουσίαση διαφορετικών σεναρίων σε ένα τεχνικό ή επιστημονικό έγγραφο, όπου ο χρήστης μπορεί να επιλέξει ποια δεδομένα ή υπολογισμούς θα συμπεριλάβει. Για παράδειγμα, σε ένα εργαλείο ανάλυσης σημάτων, τα checkboxes μπορούν να χρησιμοποιηθούν για την επιλογή μεταξύ διαφόρων τύπων φασματικών αναλύσεων ή για την εμφάνιση διαφορετικών σημάτων ταυτόχρονα. Η άμεση αλληλεπίδραση που προσφέρουν επιτρέπει την ταχύτατη σύγκριση διαφορετικών επιλογών και την προσαρμογή της ανάλυσης ανάλογα με τις ανάγκες του χρήστη.



Σχήμα 24: 2ο Παράδειγμα πλαισίου επιλογής

Από εκπαιδευτικής άποψης, τα checkboxes είναι ιδιαίτερα αποτελεσματικά καθώς επιτρέπουν στους φοιτητές να πειραματιστούν με διαφορετικά χαρακτηριστικά ή παραμέτρους χωρίς να επηρεάζουν το συνολικό σύστημα ή να αναγκάζονται να επαναφέρουν δεδομένα. Για παράδειγμα, σε ένα μάθημα που ασχολείται με την ανάλυση σημάτων, ο φοιτητής μπορεί να επιλέξει μέσω των checkboxes ποια φίλτρα θα εφαρμόσει ή ποια χαρακτηριστικά του σήματος θέλει να αναλύσει, ενώ η εφαρμογή ενημερώνει αυτόματα τα αντίστοιχα γραφήματα και αποτελέσματα.

**Ευελιξία και Διαδραστική Ανάλυση**

Η ευελιξία που προσφέρουν τα checkboxes είναι ιδιαίτερα σημαντική σε περιπτώσεις όπου χρειάζεται να γίνει ανάλυση πολλαπλών παραμέτρων ή να εφαρμοστούν διαφορετικές τεχνικές σε μια ακολουθία σημάτων. Ο χρήστης μπορεί να ελέγξει εύκολα ποιες λειτουργίες θα ενεργοποιηθούν και ποιες θα παραλειφθούν, προσαρμόζοντας τη διεργασία σύμφωνα με τις ανάγκες του χωρίς περιττές αλληλεπιδράσεις. Η χρήση τους ενθαρρύνει μια πιο ενεργή προσέγγιση στην ανάλυση δεδομένων και στην πειραματική διεργασία, καθώς ο χρήστης μπορεί να εξερευνήσει ελεύθερα διαφορετικές επιλογές.

**Επιστημονική και Εκπαιδευτική Αξία**

Η χρήση checkboxes προσθέτει σημαντική αξία τόσο στην επιστημονική τεκμηρίωση όσο και στην εκπαιδευτική διαδικασία. Σε επιστημονικά έγγραφα, οι checkboxes μπορούν να χρησιμοποιηθούν για την παρουσίαση πολύπλοκων σεναρίων, δίνοντας στους αναγνώστες τη δυνατότητα να επιλέξουν ποιες παραμέτρους ή σενάρια θα εξετάσουν. Στην εκπαιδευτική διαδικασία, προσφέρουν ένα διαδραστικό εργαλείο που ενισχύει την αφομοίωση και την κατανόηση σύνθετων εννοιών, επιτρέποντας στους φοιτητές να ενεργοποιήσουν διαφορετικές λειτουργίες με απλό και κατανοητό τρόπο.

Συνολικά, τα checkboxes αποτελούν ένα ισχυρό διαδραστικό εργαλείο που διευκολύνει την επιλογή, την ανάλυση και τη σύγκριση δεδομένων ή λειτουργιών, προσφέροντας στον χρήστη μια πιο ευέλικτη και προσαρμοσμένη εμπειρία στην επιστημονική και εκπαιδευτική ανάλυση.

### 3.4.6 Loading Animation - Χρονομετρητής (Timer)

Στη σύγχρονη ανάπτυξη λογισμικού, ιδιαίτερα σε εφαρμογές επιστημονικής οπτικοποίησης και υπολογιστικής, η εμπειρία του χρήστη παίζει καθοριστικό ρόλο. Η ενσωμάτωση διαδραστικών στοιχείων όπως το loading animation και ο χρονομετρητής βελτιώνουν σημαντικά τη διαδραστικότητα της εφαρμογής, καθιστώντας τη λειτουργία πιο κατανοητή και φιλική προς τον χρήστη.

**Loading Animation**

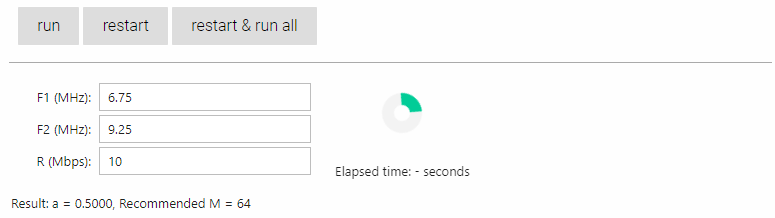
Το loading animation που χρησιμοποιήθηκε στην εφαρμογή παρέχει οπτική ανατροφοδότηση στον χρήστη, υποδεικνύοντας ότι το σύστημα επεξεργάζεται δεδομένα και εκτελεί υπολογισμούς στο παρασκήνιο. Η απουσία οπτικής ένδειξης κατά τη διάρκεια της εκτέλεσης μπορεί να προκαλέσει σύγχυση ή αβεβαιότητα σχετικά με την κατάσταση της διαδικασίας. Με την προσθήκη αυτού του οπτικού στοιχείου, ο χρήστης γνωρίζει ότι η διαδικασία βρίσκεται σε εξέλιξη, βελτιώνοντας έτσι την εμπειρία αλληλεπίδρασης.

Το animation αποτελείται από ένα κυκλικό στοιχείο που περιστρέφεται συνεχώς, προσομοιώνοντας μια διαδικασία επεξεργασίας. Το στοιχείο αυτό υλοποιήθηκε με τη χρήση CSS, όπου η περιστροφή επιτυγχάνεται με την εντολή @keyframes spin, η οποία ανανεώνει την περιστροφή του στοιχείου σε διαρκή βάση. Το χρώμα και η μορφή της κίνησης επιλέχθηκαν προσεκτικά για να δημιουργούν ένα αισθητικά ευχάριστο και ταυτόχρονα απλό animation, που δεν αποσπά την προσοχή από τον βασικό στόχο της εφαρμογής.

Ο χρονομετρητής (timer) σε ένα διαδραστικό περιβάλλον αποτελεί ένα κρίσιμο εργαλείο για την παρακολούθηση και μέτρηση του χρόνου εκτέλεσης διεργασιών ή λειτουργιών. Σε τεχνικά και επιστημονικά πλαίσια, όπως αυτά που αφορούν την ανάλυση σημάτων ή την επεξεργασία δεδομένων, η χρήση ενός timer μπορεί να παρέχει πολύτιμη πληροφόρηση για την απόδοση των αλγορίθμων, την ταχύτητα υπολογισμών, καθώς και τη βελτίωση της εμπειρίας του χρήστη, παρέχοντας του άμεση ενημέρωση για τη διάρκεια κάθε διαδικασίας.

**Διαδραστικότητα μέσω Timer**

Ο timer προσφέρει διαδραστικότητα, παρέχοντας στο χρήστη συνεχείς ενημερώσεις σχετικά με τον χρόνο που έχει παρέλθει κατά την εκτέλεση μιας διεργασίας. Αυτό είναι ιδιαίτερα χρήσιμο σε εφαρμογές όπου οι υπολογισμοί μπορεί να διαρκέσουν αρκετά, όπως η γρήγορη μεταστροφή Fourier (FFT) ή η παραγωγή μεγάλων συνόλων δεδομένων για την αναπαράσταση γραφημάτων. Η δυνατότητα του χρήστη να παρακολουθεί τον χρόνο εκτέλεσης σε πραγματικό χρόνο ενισχύει την αίσθηση ελέγχου και την κατανόηση της επίδρασης των παραμέτρων στη διάρκεια της διαδικασίας.

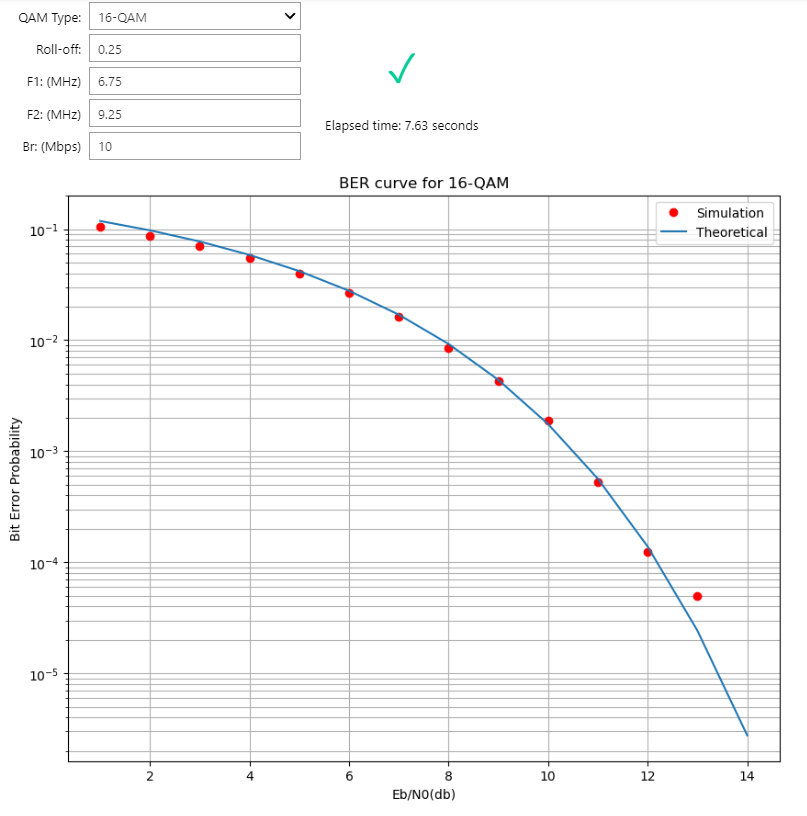


Σχήμα 25: 1ο Παράδειγμα χρονομετρητή

Στο παράδειγμα της ανάλυσης σημάτων, ο timer μπορεί να χρησιμοποιηθεί για να μετρήσει τον χρόνο που απαιτείται για την κατασκευή και την οπτικοποίηση γραφημάτων, τη φασματική ανάλυση μέσω FFT, καθώς και τη δημιουργία τυχαίου θορύβου. Η εμφάνιση του χρονικού διαστήματος σε πραγματικό χρόνο βοηθά τους χρήστες να εκτιμήσουν πόσο αποδοτικά τρέχει ο αλγόριθμος για τα δεδομένα που έχουν επιλέξει, δίνοντάς τους επίσης τη δυνατότητα να πειραματιστούν με την απόδοση του συστήματος.

**Επιστημονική Αξία του Timer**

Σε επιστημονικά περιβάλλοντα, η δυνατότητα μέτρησης χρόνου είναι εξαιρετικά σημαντική για την αξιολόγηση της απόδοσης αλγορίθμων και συστημάτων. Ο χρόνος εκτέλεσης ενός αλγορίθμου αποτελεί ένα από τα βασικά μέτρα της απόδοσής του, και η χρήση ενός timer επιτρέπει τον εντοπισμό σημείων βελτιστοποίησης ή την αξιολόγηση των επιδόσεων σε διαφορετικά σενάρια. Για παράδειγμα, όταν ένας χρήστης μεταβάλλει τις παραμέτρους ενός αλγορίθμου, μπορεί να παρατηρήσει πώς αλλάζει ο χρόνος εκτέλεσης και να αναπροσαρμόσει τις επιλογές του ώστε να επιτύχει καλύτερα αποτελέσματα.



Σχήμα 26: 2ο Παράδειγμα χρονομετρητή

Στα εκπαιδευτικά πλαίσια, η ενσωμάτωση ενός timer βοηθά τους φοιτητές να κατανοήσουν πώς η πολυπλοκότητα ενός αλγορίθμου ή η ποσότητα δεδομένων που χρησιμοποιείται μπορεί να επηρεάσει τον χρόνο εκτέλεσης. Οι φοιτητές μπορούν να παρακολουθήσουν άμεσα την επίδραση των αλλαγών στις παραμέτρους τους και να κατανοήσουν την έννοια της αποδοτικότητας υπολογισμών.

**Διαδραστική Βελτίωση της Εμπειρίας Χρήστη**

Η προσθήκη ενός timer προσφέρει στους χρήστες ένα επίπεδο διάδρασης που ενισχύει την εμπειρία τους κατά τη χρήση μιας εφαρμογής. Παρέχοντας πληροφορίες για τον χρόνο εκτέλεσης των λειτουργιών, ο χρήστης μπορεί να παρακολουθήσει τη διαδικασία σε πραγματικό χρόνο, χωρίς να χρειάζεται να υπολογίζει εκ νέου ή να βασίζεται σε μη ορατούς μηχανισμούς για να εκτιμήσει τον χρόνο που απαιτείται για την εκτέλεση ενός υπολογισμού.

Επιπλέον, ο timer μπορεί να χρησιμοποιηθεί ως μέσο για τη βελτίωση της συνολικής απόδοσης του συστήματος, παρέχοντας στον χρήστη τη δυνατότητα να εντοπίζει αργούς υπολογισμούς ή διαδικασίες που απαιτούν περισσότερη βελτιστοποίηση. Η άμεση πληροφόρηση για τον χρόνο που απομένει σε μια διαδικασία ή το συνολικό χρόνο εκτέλεσης μπορεί να βοηθήσει στην καλύτερη κατανόηση της απόδοσης της εφαρμογής και στη λήψη αποφάσεων για μελλοντικές βελτιώσεις.

Ενδεικτικό παράδειγμα κώδικα υλοποίησης Animation loader – Timer:

# Loading animation

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

loader\_html1 = widgets.HTML(

  value=loading

)

timer\_html1 = widgets.HTML(

    value="Elapsed time: - seconds"

)

**Συμπέρασμα**

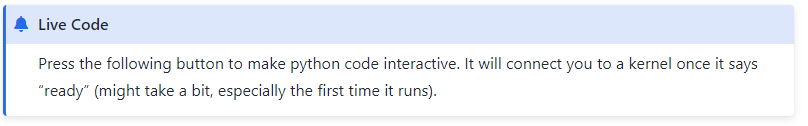
Ο timer είναι ένα εξαιρετικά χρήσιμο και διαδραστικό εργαλείο για την επιστημονική και εκπαιδευτική τεκμηρίωση. Προσφέρει την δυνατότητα μέτρησης του χρόνου εκτέλεσης σε πραγματικό χρόνο, συμβάλλοντας στη βελτίωση της εμπειρίας του χρήστη και στην ανάλυση της απόδοσης των αλγορίθμων. Στα επιστημονικά και τεχνικά πεδία, η παρακολούθηση του χρόνου είναι καίρια για την αξιολόγηση των υπολογιστικών επιδόσεων και την κατανόηση της αποδοτικότητας των μεθόδων, ενώ σε εκπαιδευτικά πλαίσια προσφέρει σημαντικές γνώσεις στους φοιτητές για τον τρόπο λειτουργίας των αλγορίθμων.

### 3.4.7 MyST Markdown

Το MyST Markdown (Markedly Structured Text) είναι μια επέκταση του τυπικού Markdown, σχεδιασμένη ειδικά για τη δημιουργία τεχνικής τεκμηρίωσης και εκπαιδευτικού υλικού μέσα σε πλατφόρμες όπως το Jupyter Book. Παρέχει πρόσθετες λειτουργίες, όπως η δυνατότητα δημιουργίας διαδραστικών στοιχείων, η ενσωμάτωση κώδικα και εξισώσεων, καθώς και η εύκολη διαχείριση υποσημειώσεων, παραπομπών και άλλων χαρακτηριστικών που δεν υποστηρίζονται στο κλασικό Markdown.

***Εκπαιδευτικά Πλεονεκτήματα του MyST Markdown***

Ένα από τα βασικά πλεονεκτήματα του MyST Markdown είναι η ικανότητά του να συνδυάζει τη δύναμη της τεκμηρίωσης με τη διαδραστικότητα του Jupyter Book. Αυτό έχει σημαντικό αντίκτυπο στη διαδικασία μάθησης και κατανόησης, επιτρέποντας στους χρήστες να:

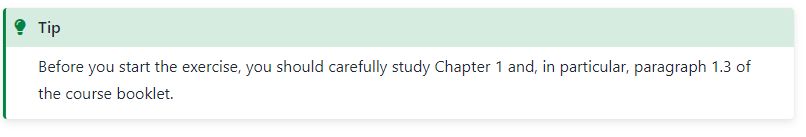
* Ενσωματώνουν Κώδικα και Εκτελέσιμα Παραδείγματα: Το MyST Markdown υποστηρίζει την ενσωμάτωση κώδικα Python και άλλων γλωσσών προγραμματισμού απευθείας στο κείμενο. Αυτό επιτρέπει στους μαθητές να εκτελούν παραδείγματα κώδικα σε πραγματικό χρόνο και να πειραματίζονται με τις παραμέτρους τους, ενισχύοντας έτσι τη διαδραστικότητα και τη hands-on προσέγγιση στη μάθηση.
* Εξισώσεις και Μαθηματικά Σύμβολα: Χρησιμοποιώντας το LaTeX και το MathJax, το MyST Markdown επιτρέπει την ενσωμάτωση σύνθετων μαθηματικών εξισώσεων, κάτι που είναι κρίσιμο για εκπαιδευτικά βιβλία σε επιστημονικά και τεχνικά πεδία. Οι εξισώσεις μπορούν να απεικονιστούν απευθείας μέσα στο κείμενο ή να αναφέρονται σε μπλοκ, κάνοντας τη θεωρητική ανάλυση πιο κατανοητή.

Σχήμα 27: Information box markdown

* Διαδραστικά Widgets: Το MyST Markdown μπορεί να ενσωματώσει διαδραστικά widgets μέσω βιβλιοθηκών όπως τα ipywidgets, προσφέροντας στους χρήστες τη δυνατότητα να αλληλεπιδρούν με τα δεδομένα. Οι μαθητές μπορούν να αλλάζουν τιμές και να βλέπουν πώς αυτές οι αλλαγές επηρεάζουν τα αποτελέσματα σε πραγματικό χρόνο, κάτι που βοηθάει ιδιαίτερα στην κατανόηση πολύπλοκων εννοιών.
* Προηγμένη Τεκμηρίωση: Το MyST Markdown επιτρέπει τη δημιουργία καλά δομημένων τεχνικών κειμένων, με δυνατότητες όπως αυτόματη δημιουργία πινάκων περιεχομένων, σημειώσεων, παραπομπών και σχολίων. Αυτό διευκολύνει τόσο τους εκπαιδευτικούς να δημιουργούν σαφή και οργανωμένα μαθήματα όσο και τους μαθητές να ακολουθούν την ύλη με ευκολία.
* Συνδέσεις με Jupyter Notebooks: Ένα σημαντικό χαρακτηριστικό του MyST είναι η δυνατότητα να ενσωματώνει και να μετατρέπει Jupyter Notebooks σε τεχνικά έγγραφα. Έτσι, οι χρήστες μπορούν να μετατρέπουν τα notebooks τους σε καλά δομημένα βιβλία ή σε άλλα εκπαιδευτικά μέσα, ενώ ταυτόχρονα διατηρούνται όλες οι δυνατότητες εκτέλεσης του κώδικα και της διαδραστικότητας.

***Επίδραση στην Κατανόηση και Μάθηση***

Η χρήση του MyST Markdown στο Jupyter Book προσφέρει έναν διαδραστικό τρόπο διδασκαλίας και μάθησης, ιδιαίτερα στα τεχνικά και επιστημονικά πεδία, όπου ο κώδικας, οι εξισώσεις και τα δεδομένα είναι βασικά εργαλεία κατανόησης. Μέσα από αυτήν τη διαδραστικότητα, οι μαθητές δεν αρκούνται σε παθητική ανάγνωση αλλά συμμετέχουν ενεργά στη διαδικασία, κατανοώντας βαθύτερα τις έννοιες και τις εφαρμογές τους.



Σχήμα 28: Tip box markdown

Η δυνατότητα να δοκιμάζουν αμέσως τις θεωρίες τους, να αλληλεπιδρούν με τα δεδομένα και να βλέπουν σε πραγματικό χρόνο τα αποτελέσματα αυξάνει την αυτονομία τους και ενισχύει την αφομοίωση της γνώσης. Αυτός ο τρόπος μάθησης όχι μόνο επιταχύνει την κατανόηση αλλά ενθαρρύνει τη δημιουργική σκέψη και την επίλυση προβλημάτων.

***Συμπεράσματα***

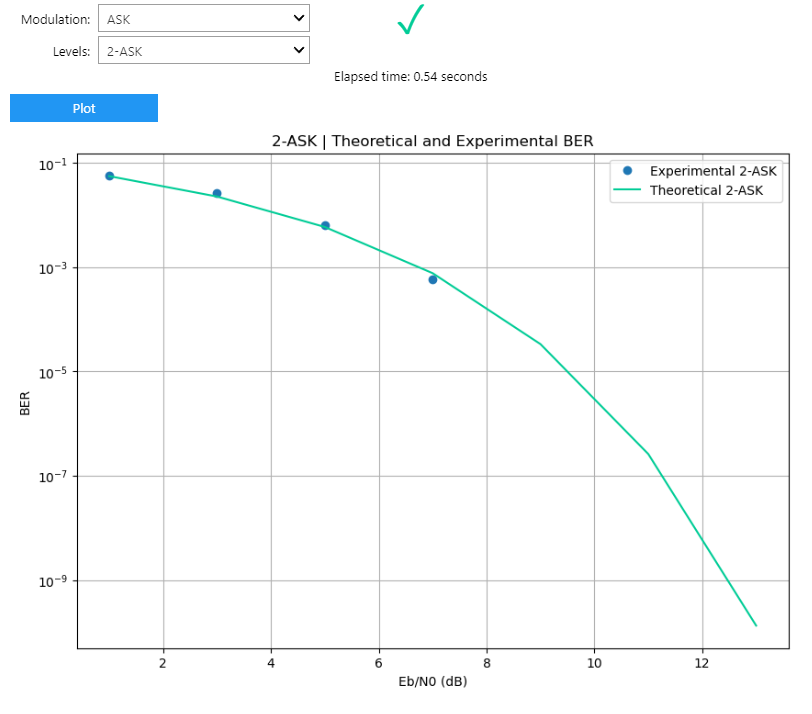
Το MyST Markdown, με τη χρήση του στο Jupyter Book, αποτελεί ένα ισχυρό εργαλείο για τη διαδραστική μάθηση και κατανόηση. Η δυνατότητα ενσωμάτωσης εκτελέσιμου κώδικα, μαθηματικών συμβόλων και διαδραστικών στοιχείων καθιστά τη διαδικασία μάθησης πιο δυναμική και ευέλικτη, ενώ η τεχνική τεκμηρίωση ενισχύεται από τις προηγμένες δυνατότητές του για δημιουργία καλά δομημένων εκπαιδευτικών υλικών.

## 3.5 Bit Error Rate Tool

Το εργαλείο Bit Error Rate (BER) Tool που αναπτύξαμε, αποτελεί ένα πολύτιμο και διαδραστικό εκπαιδευτικό εργαλείο για την ανάλυση και κατανόηση των ψηφιακών επικοινωνιών. Προσφέρει στους χρήστες τη δυνατότητα να μελετήσουν την απόδοση διαφορετικών διαμορφώσεων ψηφιακού σήματος, όπως PSK (Phase Shift Keying), ASK (Amplitude Shift Keying), FSK (Frequency Shift Keying), MSK (Minimum Shift Keying) και QAM (Quadrature Amplitude Modulation), εστιάζοντας στην απόδοση κάθε τεχνικής σε όρους BER (Bit Error Rate).

***Βασικά Χαρακτηριστικά του Εργαλείου***

Το εργαλείο αυτό προσφέρει στους χρήστες τη δυνατότητα να επιλέξουν μεταξύ των προαναφερθέντων τεχνικών διαμόρφωσης, καθώς και να προσαρμόσουν παραμέτρους όπως τα επίπεδα διαμόρφωσης (π.χ. 2-ASK, 4-PSK, 16-QAM). Καθώς ο χρήστης προσαρμόζει τις παραμέτρους, το εργαλείο παράγει τόσο θεωρητικές καμπύλες BER όσο και πειραματικά αποτελέσματα, επιτρέποντας τη σύγκριση της θεωρίας με την πρακτική.



Σχήμα 29: Bit Error Rate Tool

Στη συγκεκριμένη περίπτωση, το γράφημα 2-ASK δείχνει την αντιπαράθεση της θεωρητικής καμπύλης BER με τα πειραματικά δεδομένα, επιτρέποντας στο χρήστη να δει πώς επηρεάζεται η απόδοση της διαμόρφωσης με βάση το Eb/N0 (Energy per Bit to Noise Power Spectral Density Ratio). Αυτή η σύγκριση βοηθά στην κατανόηση της απόδοσης των διαμορφώσεων σε πραγματικά σενάρια, που μπορεί να αποκλίνουν από τα θεωρητικά αποτελέσματα λόγω παρεμβολών, θορύβου, ή άλλων παραγόντων.

***Εκπαιδευτική Ευκολία και Διαδραστικότητα***

Ένα από τα κύρια πλεονεκτήματα του εργαλείου είναι η διαδραστικότητα που προσφέρει. Ο χρήστης μπορεί εύκολα να πειραματιστεί με διαφορετικές παραμέτρους διαμόρφωσης και να δει σε πραγματικό χρόνο τα αποτελέσματα. Αυτό το χαρακτηριστικό καθιστά το εργαλείο ιδανικό για εκπαιδευτικούς σκοπούς, καθώς οι μαθητές ή φοιτητές μπορούν να πειραματιστούν και να δουν άμεσα την επίδραση που έχουν διαφορετικές τεχνικές διαμόρφωσης στο BER. Μέσω της προσαρμογής του Eb/N0 και της παρατήρησης των καμπυλών, οι χρήστες αποκτούν μια βιωματική κατανόηση του πώς διαφορετικοί παράγοντες επηρεάζουν την απόδοση του συστήματος επικοινωνίας.

***Βοήθεια στην Κατανόηση των Ψηφιακών Επικοινωνιών***

Η ικανότητα του εργαλείου να συνδυάζει θεωρητικά και πειραματικά αποτελέσματα είναι ένας εξαιρετικός τρόπος για να διευκολύνει την κατανόηση των θεμελιωδών αρχών των ψηφιακών επικοινωνιών. Παρέχοντας άμεση οπτική ανατροφοδότηση για την απόδοση κάθε διαμόρφωσης, το εργαλείο επιτρέπει στους χρήστες να εξετάσουν τις βασικές αρχές χωρίς την ανάγκη για εξειδικευμένη γνώση προγραμματισμού ή πολύπλοκων μαθηματικών υπολογισμών. Επιπλέον, η σύγκριση των αποτελεσμάτων σε διαφορετικές συνθήκες βοηθάει να γίνει κατανοητό το πώς το θόρυβος και άλλοι παράγοντες επηρεάζουν την ποιότητα του σήματος και την αξιοπιστία των επικοινωνιών.

***Συμπέρασμα***

Το εργαλείο Bit Error Rate Tool αποτελεί μια εξαιρετική εκπαιδευτική πλατφόρμα, ειδικά για την κατανόηση της απόδοσης διαφορετικών διαμορφώσεων σήματος και τη σχέση τους με τον ρυθμό σφαλμάτων bit (BER). Μέσω της διαδραστικής και άμεσης οπτικοποίησης, οι χρήστες μπορούν να αποκτήσουν μια ολοκληρωμένη εικόνα για τις τεχνικές διαμόρφωσης που χρησιμοποιούνται στις σύγχρονες ψηφιακές επικοινωνίες, καθιστώντας το ένα πολύτιμο εργαλείο για φοιτητές, ερευνητές και επαγγελματίες στον τομέα.

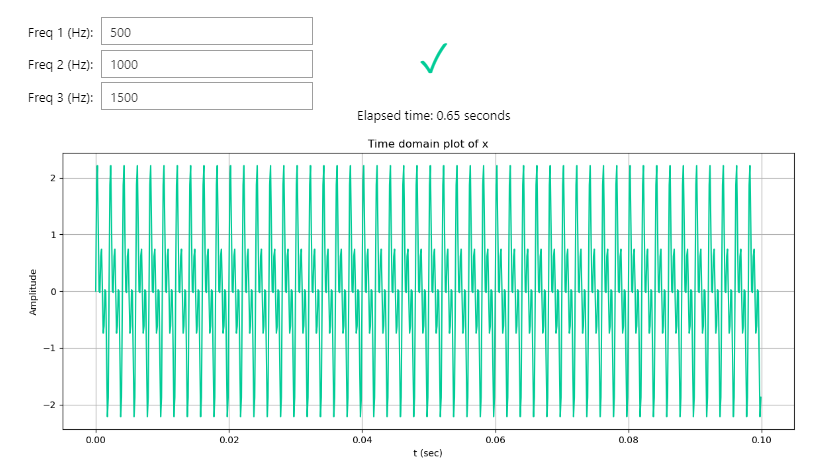
Κεφάλαιο 4

# Αποτελέσματα

## 4.1 Lab Exercise 1: Εισαγωγή στα Συστήματα Ψηφιακής Μετάδοσης

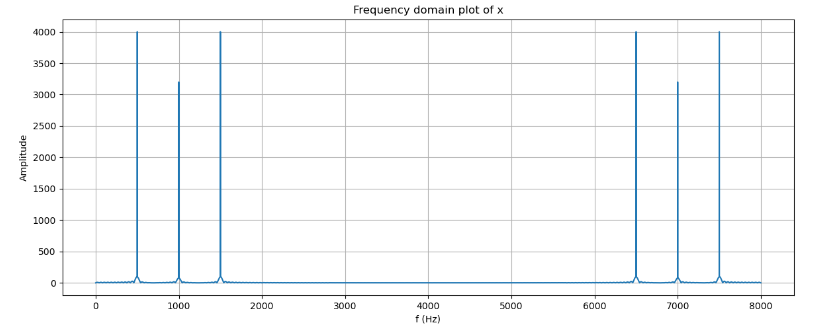
### 4.1.1 Αποτελέσματα Προσομοίωσης

Οι προσομοιώσεις που παρουσιάζονται καλύπτουν διαφορετικές καταστάσεις σήματος, με ποικίλες συχνότητες και επίπεδα θορύβου. Σε κάθε προσομοίωση παρατηρείται η ανάλυση του σήματος τόσο στον χρονικό όσο και στον συχνοτικό τομέα, αναδεικνύοντας τις κύριες συχνότητες που εισήχθησαν, καθώς και την επίδραση της δειγματοληψίας και του προστιθέμενου θορύβου.



Σχήμα 30: Αναπαράσταση σήματος με είσοδο 3 συχνοτήτων

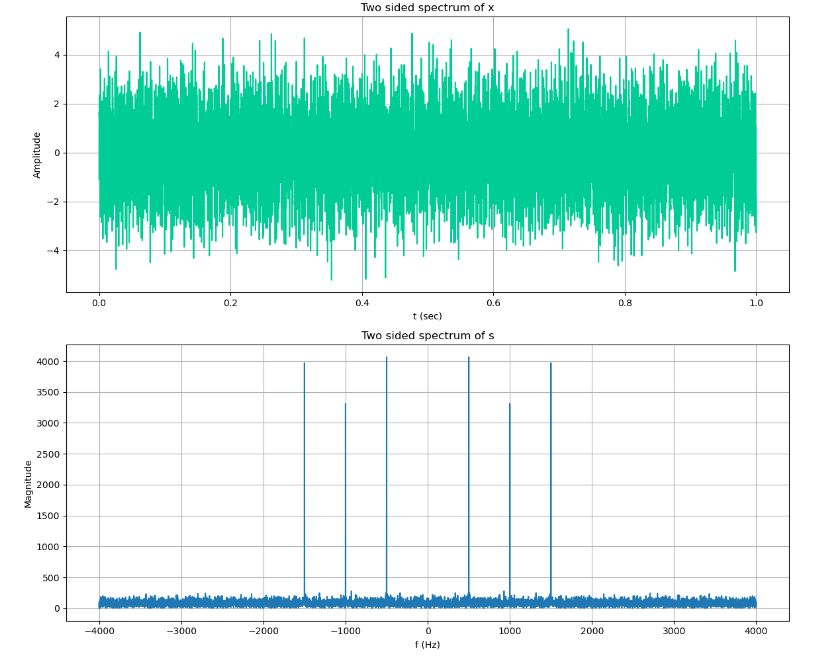
Στις περιπτώσεις όπου δεν προστέθηκε θόρυβος, οι συχνότητες των σημάτων είναι απόλυτα διακριτές και το φάσμα παρουσιάζει σαφείς κορυφές στις αντίστοιχες συχνότητες. Όταν ο θόρυβος προστίθεται (είτε εσκεμμένα, όπως στην τελευταία περίπτωση), οι κορυφές των συχνοτήτων είναι μεν ακόμα ανιχνεύσιμες, αλλά επηρεάζονται από την παρουσία θορύβου, που οδηγεί σε χαμηλότερη διακριτότητα.



Σχήμα 31: Αναπαράσταση φάσματος σήματος

### 4.1.2 Ανάλυση Επιδόσεων

Οι προσομοιώσεις αποδεικνύουν τη δυνατότητα ανίχνευσης και ανάλυσης συχνοτήτων με ακρίβεια υπό διαφορετικές συνθήκες. Στις περιπτώσεις χωρίς θόρυβο, οι συχνότητες είναι απόλυτα ευκρινείς, με άριστη απόδοση της ανάλυσης φάσματος, καθώς οι συχνότητες εισόδου αντικατοπτρίζονται άμεσα στο φάσμα. Επιπλέον, ο χρόνος εκτέλεσης σε κάθε προσομοίωση δείχνει ότι οι μέθοδοι είναι αρκετά αποδοτικές και μπορούν να παράγουν αποτελέσματα σε μικρό χρονικό διάστημα, ακόμα και σε περιβάλλοντα όπου προστίθεται θόρυβος.



Σχήμα 32: Αναπαράσταση σήματος και φάσματος αυτού με προσθήκη θορύβου

Η προσθήκη θορύβου αυξάνει την πολυπλοκότητα της ανάλυσης, καθώς επηρεάζει την καθαρότητα του σήματος. Ωστόσο, ακόμα και υπό αυτές τις συνθήκες, το σήμα παραμένει ανιχνεύσιμο, κάτι που αποδεικνύει την ικανότητα της προσομοίωσης να ανταπεξέρχεται σε ρεαλιστικά σενάρια ψηφιακής μετάδοσης.

### 4.1.3 Αδυναμίες και Προκλήσεις

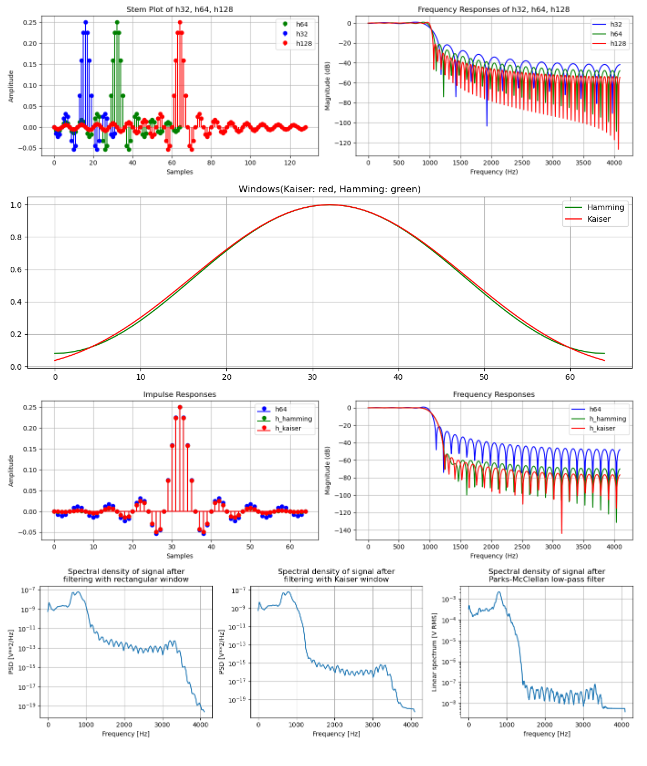
Επιπλέον, οι περιορισμοί στη συχνότητα δειγματοληψίας είναι εμφανείς, καθώς οι συχνότητες δεν μπορούν να υπερβούν ένα συγκεκριμένο όριο (π.χ. 500 Hz στις συγκεκριμένες προσομοιώσεις). Αυτός ο περιορισμός υπαγορεύεται από τον κανόνα Nyquist, και η υπέρβαση του θα οδηγούσε σε αναδίπλωση συχνοτήτων (aliasing), καθιστώντας την ανάλυση ανακριβή.

Εν κατακλείδι, οι προσομοιώσεις αυτές προσφέρουν μια εξαιρετική βάση για την κατανόηση της συμπεριφοράς των σημάτων στον χρονικό και συχνοτικό τομέα, καθώς και την επίδραση του θορύβου και της δειγματοληψίας στην ανάλυση του σήματος.

## 4.2 Lab Exercise 2: Ψηφιακά Φίλτρα

### 4.2.1 Αποτελέσματα Προσομοίωσης

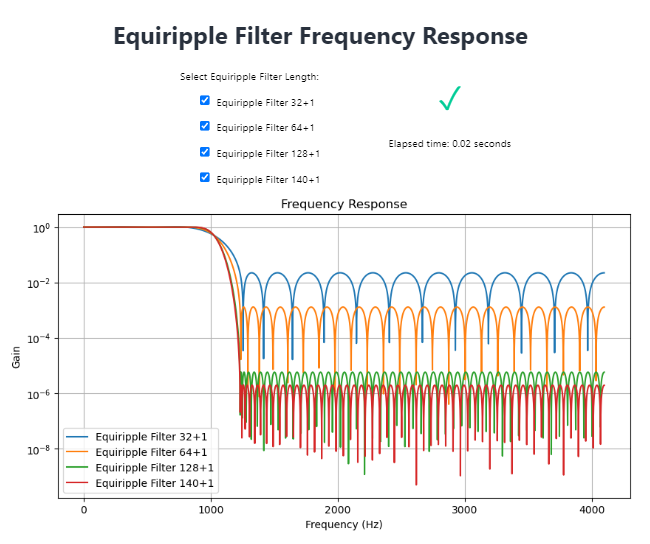
Στην παρούσα μελέτη, αναλύθηκαν τα αποτελέσματα προσομοιώσεων που αφορούν την εφαρμογή και αξιολόγηση φίλτρων ψηφιακών σημάτων. Οι προσομοιώσεις πραγματοποιήθηκαν για να εξεταστεί η απόκριση των φίλτρων σε διάφορες συνθήκες και να αξιολογηθεί η ακρίβεια των αποτελεσμάτων σε σχέση με τα θεωρητικά αναμενόμενα. Τα διαγράμματα που παρουσιάζονται περιλαμβάνουν την απόκριση παλμού και συχνότητας, τη φασματική πυκνότητα ισχύος, καθώς και τη συγκριτική ανάλυση φίλτρων με διαφορετικά παραθύρα και διαφορετικά μήκη.



Σχήμα 33: Ψηφιακά Φίλτρα

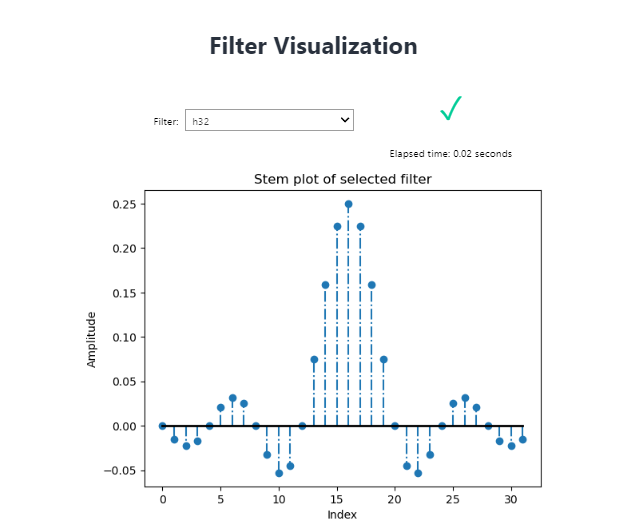
### 4.2.2 Ανάλυση Επιδόσεων

Τα αποτελέσματα των προσομοιώσεων δείχνουν μια σαφή ταύτιση με τις θεωρητικές προσδοκίες σε μεγάλο βαθμό. Συγκεκριμένα, η απόκριση των φίλτρων bandpass τόσο στον χρονικό όσο και στον συχνοτικό τομέα είναι εξαιρετικά κοντά σε ό,τι αναμένεται θεωρητικά. Η αποκοπή των ανεπιθύμητων συχνοτήτων πραγματοποιείται με ακρίβεια, και η περιοχή διέλευσης παρουσιάζει ελάχιστες αποκλίσεις, οι οποίες μπορούν να θεωρηθούν αμελητέες για τις περισσότερες εφαρμογές ψηφιακής μετάδοσης.

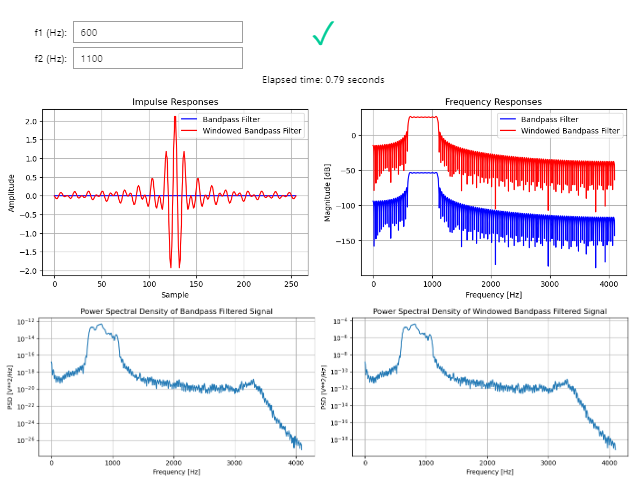


Σχήμα 34: Απόκριση συχνότητας Equiripple φίλτρου

Η απόκριση των equiripple φίλτρων, όπως φαίνεται από τις προσομοιώσεις, ανταποκρίνεται επίσης στις θεωρητικές προβλέψεις, με τα φίλτρα μεγαλύτερου μήκους να προσφέρουν καλύτερη απόδοση στη ζώνη αποκοπής και να μειώνουν τις ταλαντώσεις στη ζώνη διέλευσης. Οι ταλαντώσεις που παρατηρούνται είναι εντός των αναμενόμενων ορίων για φίλτρα αυτού του τύπου. Επιπλέον, τα φίλτρα με παράθυρο Kaiser παρουσιάζουν μικρότερες αποκλίσεις στη φασματική πυκνότητα ισχύος, γεγονός που συμφωνεί με τη θεωρία ότι αυτό το παράθυρο προσφέρει καλύτερη απόδοση αποκοπής σε σύγκριση με το παράθυρο Hamming.

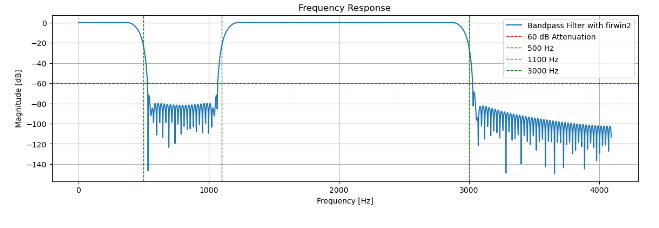


Σχήμα 35: Οπτικοποίηση Φίλτρου

Ο χρόνος εκτέλεσης των προσομοιώσεων είναι εξαιρετικά σύντομος, επιτρέποντας γρήγορη επεξεργασία και παραγωγή αποτελεσμάτων. Αυτό υποδηλώνει ότι τα χρησιμοποιούμενα φίλτρα είναι ιδιαίτερα αποδοτικά για χρήση σε πραγματικό χρόνο ή σε συνθήκες όπου απαιτείται γρήγορη επεξεργασία ψηφιακών σημάτων.

Σχήμα 36: Παράδειγμα ζωνοπερατού φίλτρου

### 4.2.3 Αδυναμίες και Προκλήσεις

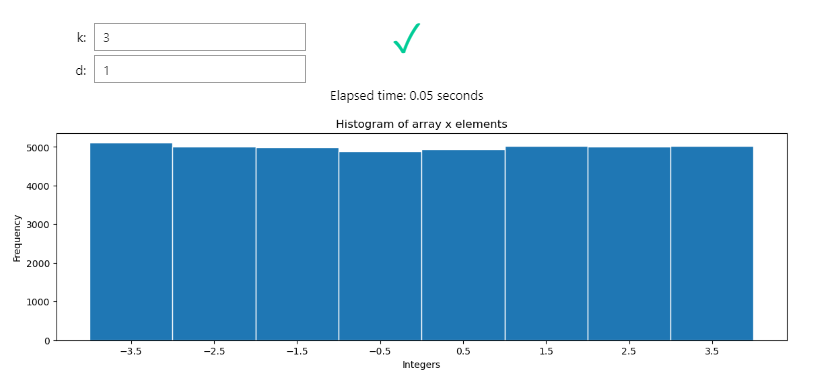
Δεν παρατηρούνται αξιοσημείωτες αδυναμίες στην προσομοίωση αυτή. Η μικρή απόκλιση σε πολύ χαμηλές τιμές Eb/N0 οφείλεται σε τυχαίο θόρυβο και είναι αναμενόμενη.

Σχήμα 37: Παράδειγμα φίλτρου με 2 ζώνες διέλευσης

## 4.3 Lab Exercise 3: Συγχρονισμένα Φίλτρα και Διαμόρφωση L-ASK

### 4.3.1 Αποτελέσματα Προσομοίωσης

Τα διαγράμματα που παρουσιάζονται αφορούν την απόδοση των συστημάτων 4-ASK σε συνθήκες με προσθήκη θορύβου, όπου μελετάται η αναλογία bit σφαλμάτων (BER) σε σχέση με την ενέργεια ανά bit προς το φάσμα θορύβου (Eb/No). Η θεωρητική καμπύλη BER συγκρίνεται με την πειραματική, ενώ επιπλέον προσομοιώσεις περιλαμβάνουν τον σχεδιασμό και αξιολόγηση διαμορφώσεων και φίλτρων ψηφιακών σημάτων. Οι πίνακες με τα ορθογώνια και ημιτονοειδή σήματα, όπως και η σύγκριση των BER σε διαφορετικές συνθήκες Eb/No, παρέχουν κρίσιμα δεδομένα για την ανάλυση της απόδοσης των συστημάτων.

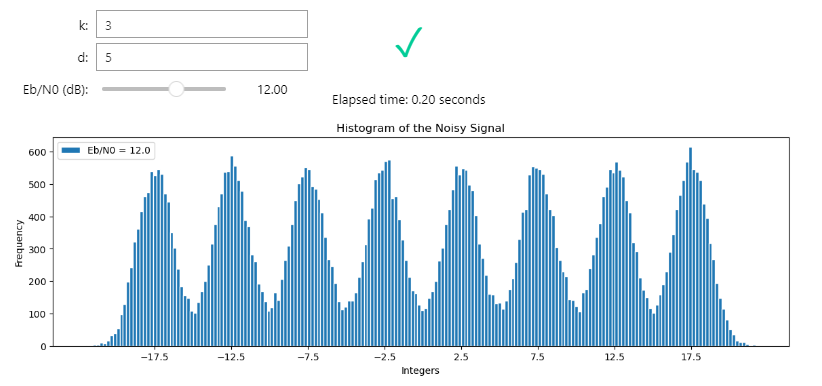


Σχήμα 38: Ιστόγραμμα πίνακα x στοιχείων

### 4.3.2 Ανάλυση Επιδόσεων

Τα αποτελέσματα των προσομοιώσεων είναι σε πολύ καλή συμφωνία με τις θεωρητικές προσδοκίες. Η καμπύλη του πειραματικού BER πλησιάζει πολύ την θεωρητική καμπύλη, ιδιαίτερα στις χαμηλότερες τιμές του Eb/No. Όσο το Eb/No αυξάνεται, οι αποκλίσεις μειώνονται περαιτέρω, επιβεβαιώνοντας ότι το σύστημα λειτουργεί σωστά σε συνθήκες αυξημένης ποιότητας σήματος. Οι μικρές αποκλίσεις που παρατηρούνται είναι αναμενόμενες λόγω της επίδρασης του θορύβου και άλλων παραγόντων που μπορούν να επηρεάσουν την πραγματική μετάδοση του σήματος.

Η ορθογωνική και ημιτονοειδής απεικόνιση των σημάτων και οι αντίστοιχες προσομοιώσεις δείχνουν πως οι συνθήκες διαμόρφωσης και αποδιαμόρφωσης είναι σωστές. Η δομή των σημάτων όπως φαίνεται στα διαγράμματα παραπέμπει σε κλασικά αποτελέσματα που συναντώνται στη θεωρία της ψηφιακής διαμόρφωσης. Οι διαφορές μεταξύ των διαφορετικών τιμών Eb/No που παρατηρούνται στα διαγράμματα BER είναι απόλυτα ευθυγραμμισμένες με τις θεωρητικές προσδοκίες, αποδεικνύοντας την ορθότητα της προσομοίωσης.

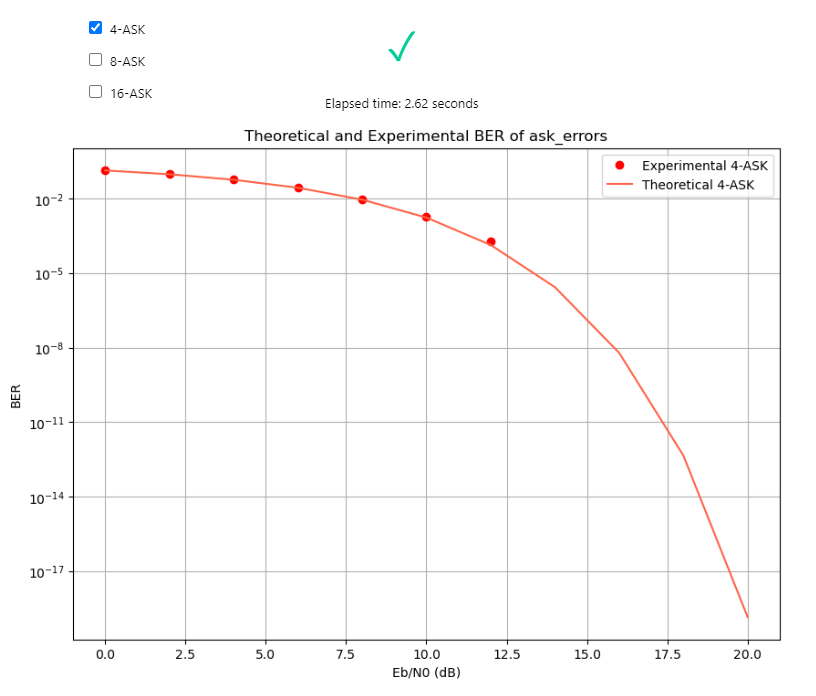


Σχήμα 39: Ιστόγραμμα σήματος με θόρυβο

Η απόδοση των προσομοιώσεων ως προς τον χρόνο εκτέλεσης είναι επίσης ένα σημαντικό στοιχείο που πρέπει να ληφθεί υπόψη. Στα διαγράμματα που παρουσιάστηκαν, ο χρόνος εκτέλεσης των προσομοιώσεων κυμαίνεται από 0.05 έως 2.62 δευτερόλεπτα, ανάλογα με την πολυπλοκότητα της κάθε προσομοίωσης. Αυτό το εύρος χρόνου εκτέλεσης θεωρείται πολύ αποδοτικό για την προσομοίωση συστημάτων διαμόρφωσης ψηφιακών σημάτων, όπως τα συστήματα 4-ASK που χρησιμοποιούνται σε αυτές τις προσομοιώσεις.

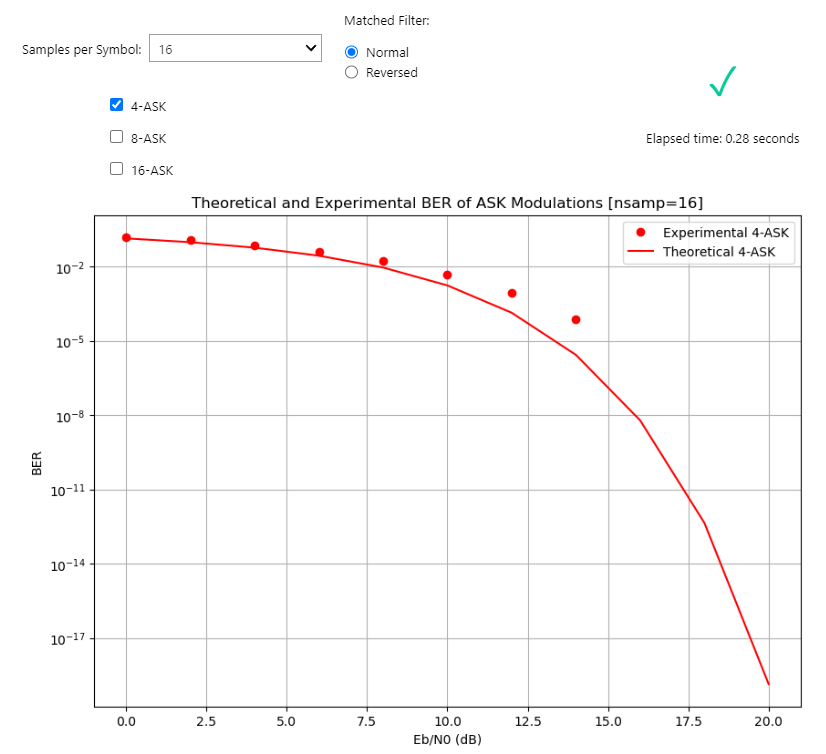
Ο μικρός χρόνος εκτέλεσης, ιδιαίτερα για τις προσομοιώσεις μικρότερης πολυπλοκότητας (π.χ. χωρίς θόρυβο ή με λιγότερα δείγματα), δείχνει την αποδοτική υλοποίηση των αλγορίθμων και την καλή διαχείριση των υπολογιστικών πόρων. Για παράδειγμα, προσομοιώσεις όπως οι απεικονίσεις με το nsamp=8 ολοκληρώνονται σε λιγότερο από 0.2 δευτερόλεπτα, γεγονός που υποδεικνύει ότι οι υλοποιημένοι αλγόριθμοι είναι κατάλληλοι για ταχεία ανάλυση σε εργαστηριακό περιβάλλον.

Ωστόσο, σε περιπτώσεις όπου προστίθεται θόρυβος ή αυξάνεται ο αριθμός των δειγμάτων, ο χρόνος εκτέλεσης αυξάνεται, όπως φαίνεται σε προσομοιώσεις με χρόνο 2.62 δευτερόλεπτα. Παρόλο που ο χρόνος αυτός εξακολουθεί να είναι αποδεκτός, η αύξηση αυτή μπορεί να αποδοθεί στην ανάγκη εκτενέστερων υπολογισμών για την προσομοίωση θορύβου και για την ακριβέστερη ανάλυση των bit σφαλμάτων (BER).



Σχήμα 40: BER L - ASK διαμόρφωσης

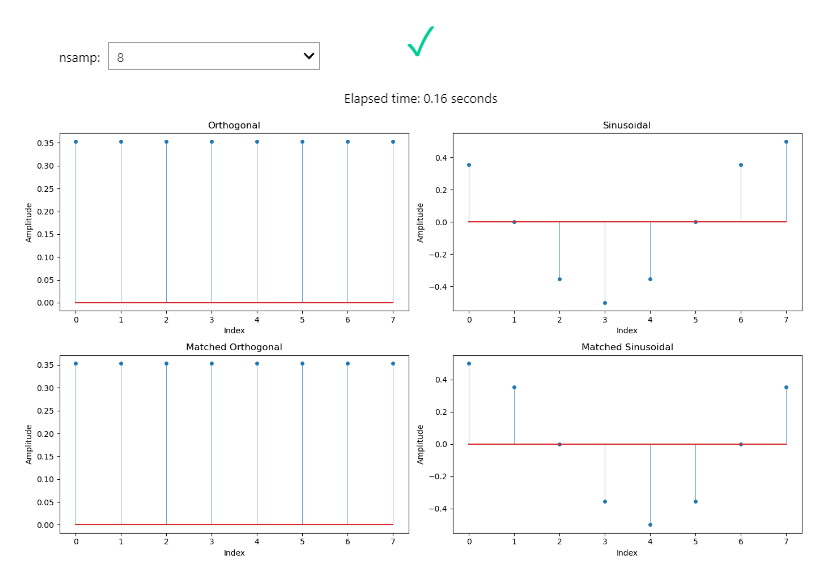
Συνολικά, η απόδοση των προσομοιώσεων ως προς τον χρόνο εκτέλεσης είναι πολύ καλή και επιτρέπει γρήγορες και αποδοτικές προσομοιώσεις, ακόμα και σε περιβάλλοντα με αυξημένες απαιτήσεις υπολογιστικής ισχύος. Αυτό καθιστά τις προσομοιώσεις κατάλληλες για εργαστηριακές εφαρμογές και μελλοντική ενσωμάτωση σε πραγματικά συστήματα.



Σχήμα 41: BER L - ASK διαμόρφωσης με/χωρίς αναστροφή πίνακα

### 4.3.3 Αδυναμίες και Προκλήσεις

Παρότι οι προσομοιώσεις είναι σε μεγάλο βαθμό ακριβείς, υπάρχουν μερικές αδυναμίες και προκλήσεις που μπορούν να αναφερθούν. Πρώτον, οι αποκλίσεις μεταξύ πειραματικών και θεωρητικών τιμών BER στις χαμηλές τιμές Eb/No είναι ελαφρώς αυξημένες. Αυτό πιθανώς οφείλεται στην τυχαία φύση του θορύβου και στις παραδοχές που γίνονται στην προσομοίωση. Μια περαιτέρω προσέγγιση θα ήταν η αύξηση του αριθμού δειγμάτων για να μειωθούν οι στατιστικές αποκλίσεις και να προσεγγιστεί καλύτερα η θεωρητική καμπύλη.



Σχήμα 42: Αναπάρασταση ορθογωνικού και ημιτονοειδούς παλμού με nsamp σημεία

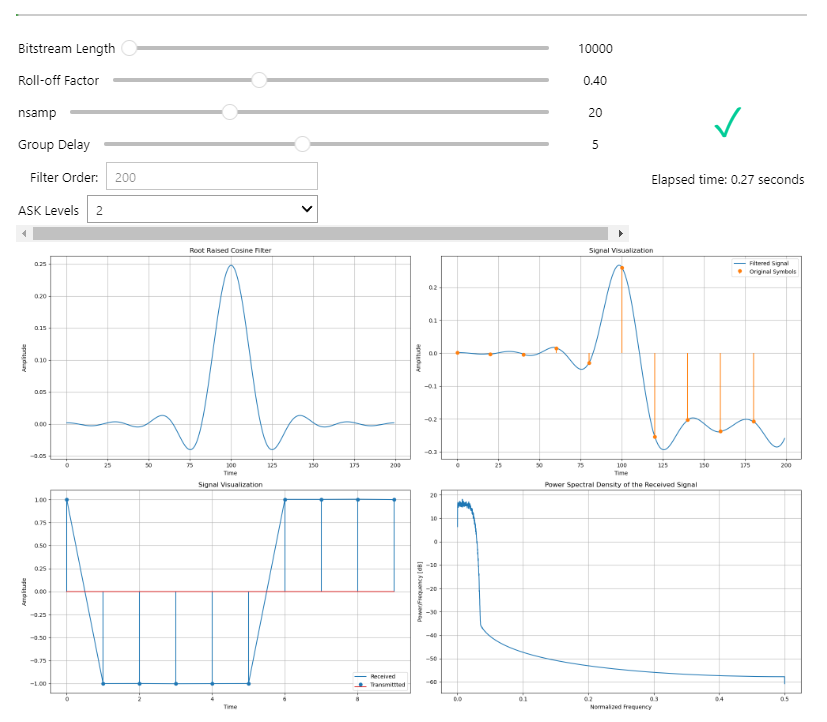
Συνοψίζοντας, τα αποτελέσματα των προσομοιώσεων επιβεβαιώνουν τη θεωρητική απόδοση των συστημάτων ψηφιακής μετάδοσης που χρησιμοποιούν διαμόρφωση ASK, με μικρές αποκλίσεις που μπορούν να αποδοθούν στις συνθήκες θορύβου και την πολυπλοκότητα της υλοποίησης.

## 4.4 Lab Exercise 4: Σηματοδότηση Nyquist και Διαμόρφωση L-ASK

### 4.4.1 Αποτελέσματα Προσομοίωσης

Τα διαγράμματα αποδεικνύουν εξαιρετική ακρίβεια στην αποτύπωση της επίδοσης των διαφορετικών κωδικοποιήσεων (Gray και Natural) σε συνάρτηση με το Eb/N0. Τα αποτελέσματα είναι συνεπή με τις θεωρητικές προβλέψεις για την κωδικοποίηση Gray, η οποία παρουσιάζει χαμηλότερα σφάλματα σε σχέση με την κωδικοποίηση Natural, όπως αναμενόταν. Η απόδοση της προσομοίωσης με Gray κωδικοποίηση είναι κοντά στη θεωρητική, γεγονός που καταδεικνύει την αξία της για την ελαχιστοποίηση του ρυθμού σφαλμάτων.

Η προσομοίωση της Natural κωδικοποίησης εμφανίζει σαφώς υψηλότερο ρυθμό σφαλμάτων (BER) σε σύγκριση με την Gray, με μια ξεκάθαρη απόκλιση από τη θεωρητική καμπύλη. Αυτή η απόκλιση ήταν αναμενόμενη, όπως αναφέρθηκε, λόγω της εγγενούς αδυναμίας της Natural κωδικοποίησης να διαχειριστεί επαρκώς γειτονικά σύμβολα.

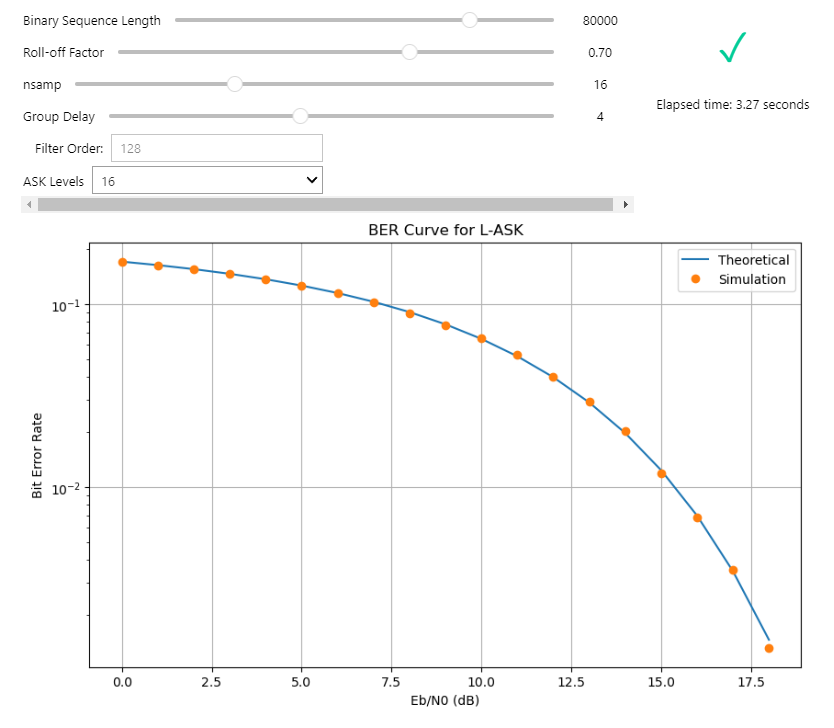


Σχήμα 43: Δημιουργία σήματος με Nyquist φίλτρα  
 (Χρονική και φασματική αναπαράσταση)

Συνολικά, η αξιοπιστία των προσομοιώσεων είναι πολύ υψηλή, με την Gray κωδικοποίηση να παρουσιάζει σχεδόν τέλεια συμφωνία με τις θεωρητικές καμπύλες, ενώ η Natural έχει προβλέψιμες αποκλίσεις.

### 4.4.2 Ανάλυση Επιδόσεων

Οι χρόνοι που εμφανίζονται στα στιγμιότυπα (3.27 δευτερόλεπτα στο δεύτερο διάγραμμα και 8.13 δευτερόλεπτα στο πρώτο) δείχνουν ότι η ταχύτητα προσομοίωσης είναι αρκετά ικανοποιητική για τον όγκο δεδομένων που επεξεργάζεται. Τα διαγράμματα δημιουργούνται σε πραγματικό χρόνο, με τις τιμές να ενημερώνονται άμεσα καθώς αλλάζουν οι παράμετροι, όπως το μήκος της ακολουθίας και οι ρυθμίσεις διαμόρφωσης. Αυτή η απόκριση θεωρείται επαρκώς γρήγορη για εφαρμογές εργαστηριακού επιπέδου και ανάλυσης συστημάτων επικοινωνίας.



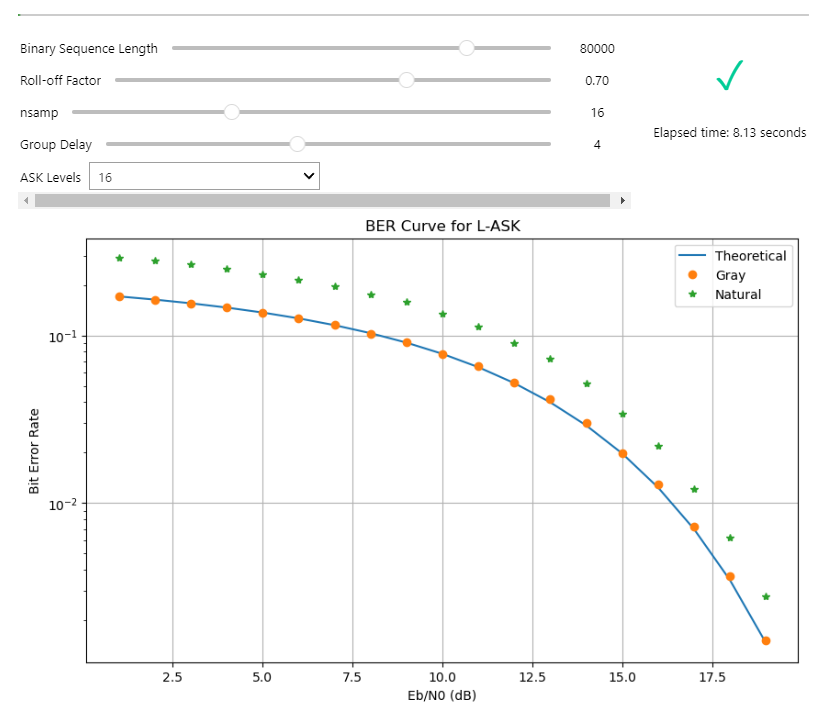
Σχήμα 44: BER L - ASK διαμόρφωση

Σε υψηλότερες τιμές για παραμέτρους όπως το μήκος του bitstream ή το πλήθος δειγμάτων ανά σύμβολο, μπορεί να εμφανιστεί ελαφριά επιβράδυνση στον χρόνο απόκρισης, αλλά αυτό είναι αναμενόμενο λόγω της αυξημένης υπολογιστικής πολυπλοκότητας. Συνολικά, η ταχύτητα προσομοίωσης θεωρείται πολύ ικανοποιητική για ακαδημαϊκή και ερευνητική χρήση.

### 4.4.3 Αδυναμίες και Προκλήσεις

Ο χρόνος της προσομοίωσης ορισμένων διαγραμμάτων είναι σημαντικά μεγαλύτερος σε σύγκριση με προηγούμενες προσομοιώσεις. Αυτό οφείλεται πιθανότατα στην αυξημένη πολυπλοκότητα των παραμέτρων της προσομοίωσης, όπως το μεγάλο μήκος της δυαδικής ακολουθίας (Binary Sequence Length), η τιμή του Roll-off Factor και ο αριθμός των δειγμάτων ανά σύμβολο (nsamp). Το μέγεθος των δεδομένων προς επεξεργασία αυξάνεται σημαντικά, καθώς και οι υπολογιστικές απαιτήσεις της προσομοίωσης για την απεικόνιση των τριών διαφορετικών καμπυλών (Theoretical, Gray, και Natural).

Η χρονική διάρκεια των 25.09 δευτερολέπτων είναι αναμενόμενη και αποδεκτή για μια τόσο λεπτομερή προσομοίωση, ειδικά με την εισαγωγή περισσότερων μεταβλητών και το γεγονός ότι υπολογίζεται η απόδοση για διαφορετικές μορφές κωδικοποίησης. Ωστόσο, υπάρχουν περιθώρια βελτιστοποίησης, ενδεχομένως μέσω βελτίωσης του αλγορίθμου ή της χρήσης πιο αποδοτικών τεχνικών για τον περιορισμό της υπολογιστικής πολυπλοκότητας.Η συνολική επίδοση της προσομοίωσης και η ταχύτητα απόκρισης κρίνονται πολύ ικανοποιητικές για το εύρος των δεδομένων και των παραμέτρων που επεξεργάζονται. Η αδυναμία της Natural κωδικοποίησης είναι σαφής και αναμενόμενη, αναδεικνύοντας έτσι την ανωτερότητα της Gray κωδικοποίησης στη μείωση των σφαλμάτων.

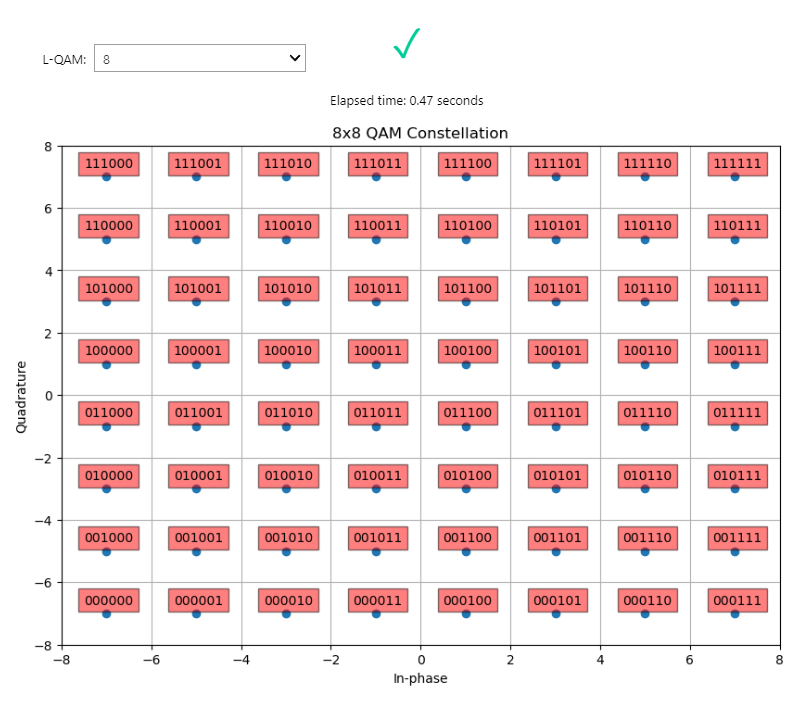


Σχήμα 45:BER L – ASK (με Gray vs Natural Coding)

## 4.5 Lab Exercise 5: Διαμόρφωση QAM και PSK

### 4.5.1 Αποτελέσματα Προσομοίωσης

Η παρούσα ανάλυση αφορά την απόδοση συστημάτων ψηφιακής διαμόρφωσης, όπως η 4-PSK (QPSK), 16-QAM, 16-PSK και 64-QAM, με έμφαση στη σύγκριση των προσομοιωμένων καμπυλών Ρυθμού Σφαλμάτων Bit (BER) με τις αντίστοιχες θεωρητικές καμπύλες. Τα αποτελέσματα των προσομοιώσεων αποτυπώνουν τη συνολική απόδοση κάθε συστήματος και εξετάζουν την ταχύτητα με την οποία παράγονται τα διαγράμματα, καθώς και πιθανές αδυναμίες στην προσομοίωση.

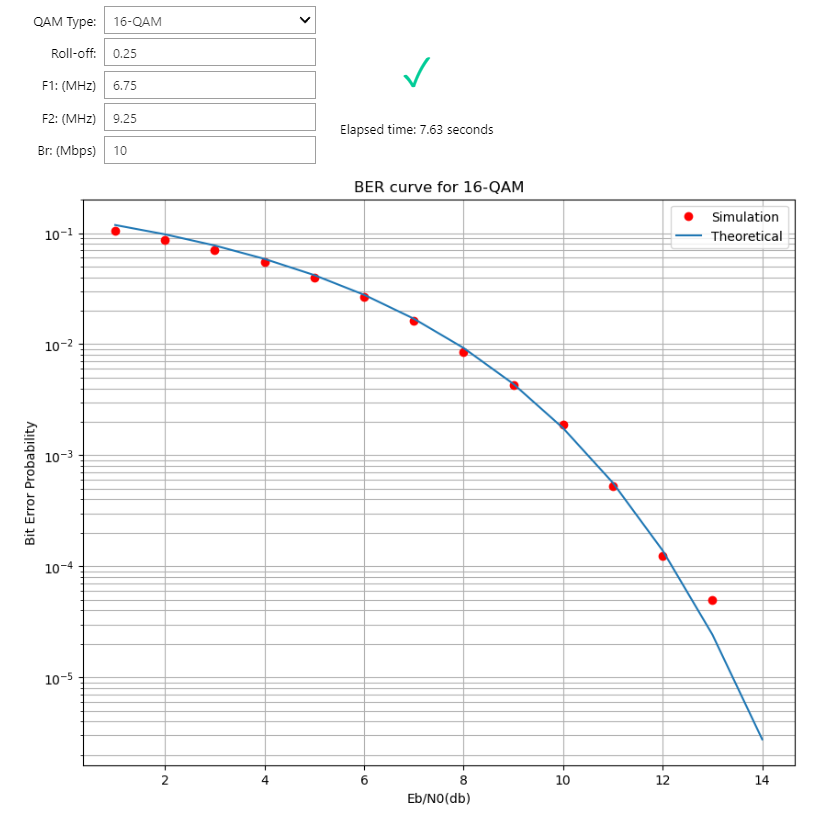


Σχήμα 46Q Αστερισμός MxM QAM

### 4.5.2 Ανάλυση Επιδόσεων

Η απεικόνιση του αστερισμού (constellation diagram) για το σχήμα διαμόρφωσης 16-PSK είναι ιδιαίτερα ευκρινής και ακριβής. Τα σύμβολα τοποθετούνται συμμετρικά στον χώρο in-phase και quadrature, όπως απαιτείται από τη διαμόρφωση PSK. Ο χρόνος απόκρισης είναι μόλις 0.15 δευτερόλεπτα, αποδεικνύοντας την ταχύτητα της διαδικασίας.

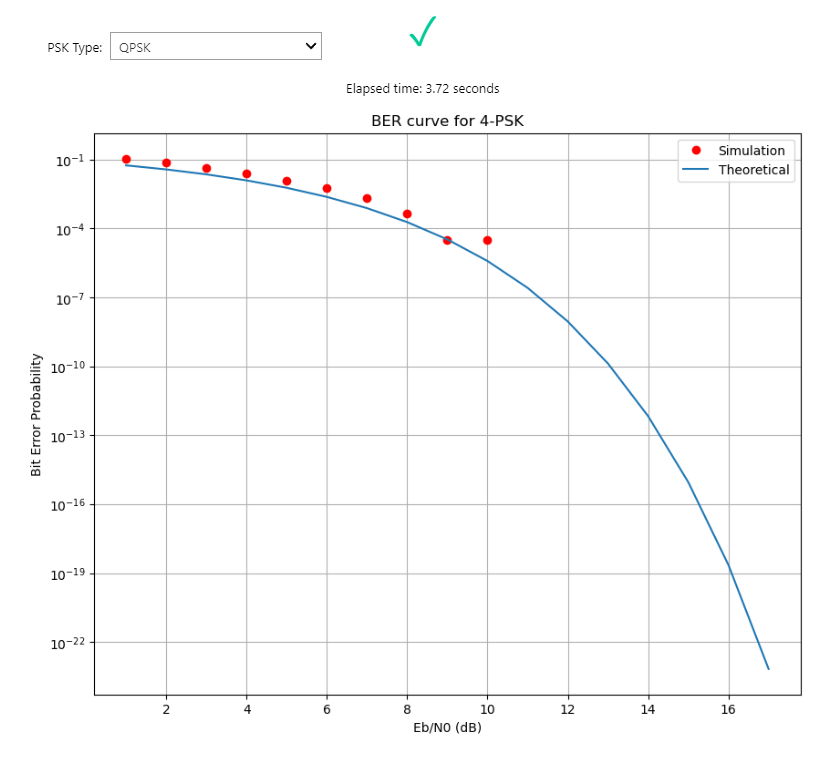
Η αναπαράσταση του αστερισμού για το σχήμα 8x8 QAM (64-QAM) παρουσιάζει την πλήρη και σωστή τοποθέτηση των συμβόλων στο σύστημα in-phase και quadrature. Η απεικόνιση είναι καθαρή, και οι θέσεις των συμβόλων διαχωρίζονται με σαφήνεια, δείχνοντας την ακρίβεια της διαμόρφωσης. Ο χρόνος απεικόνισης, στα 0.47 δευτερόλεπτα, είναι γρήγορος, καθιστώντας το σύστημα κατάλληλο για εργαστηριακές μετρήσεις και ανάλυση.



Σχήμα 47: BER QAM

Η προσομοίωση για το σχήμα διαμόρφωσης 16-QAM εμφανίζει πολύ καλή αντιστοιχία με τη θεωρητική καμπύλη. Ο ρυθμός σφαλμάτων για το 16-QAM ακολουθεί τη θεωρία σε όλο το εύρος του Eb/N0, με ελάχιστες αποκλίσεις που δεν επηρεάζουν την συνολική αξιοπιστία του μοντέλου. Ο χρόνος προσομοίωσης, αν και μεγαλύτερος από την περίπτωση της QPSK, είναι 7.63 δευτερόλεπτα, παραμένοντας εντός των επιθυμητών ορίων.

Η σύγκριση μεταξύ της θεωρητικής καμπύλης και της προσομοίωσης για την 4-PSK διαμόρφωση (QPSK) δείχνει μια **άριστη αντιστοιχία**. Οι προσομοιωμένες τιμές για τον ρυθμό σφαλμάτων bit (BER) βρίσκονται σε πολύ μικρή απόσταση από τη θεωρητική καμπύλη, κάτι που αποδεικνύει την **ακρίβεια της προσομοίωσης** σε όλες τις τιμές του λόγου Eb/N0. Ο χρόνος προσομοίωσης είναι μόλις **3.72 δευτερόλεπτα**, καθιστώντας τη διαδικασία εξαιρετικά **αποτελεσματική**.



Σχήμα 48: BER M - PSK

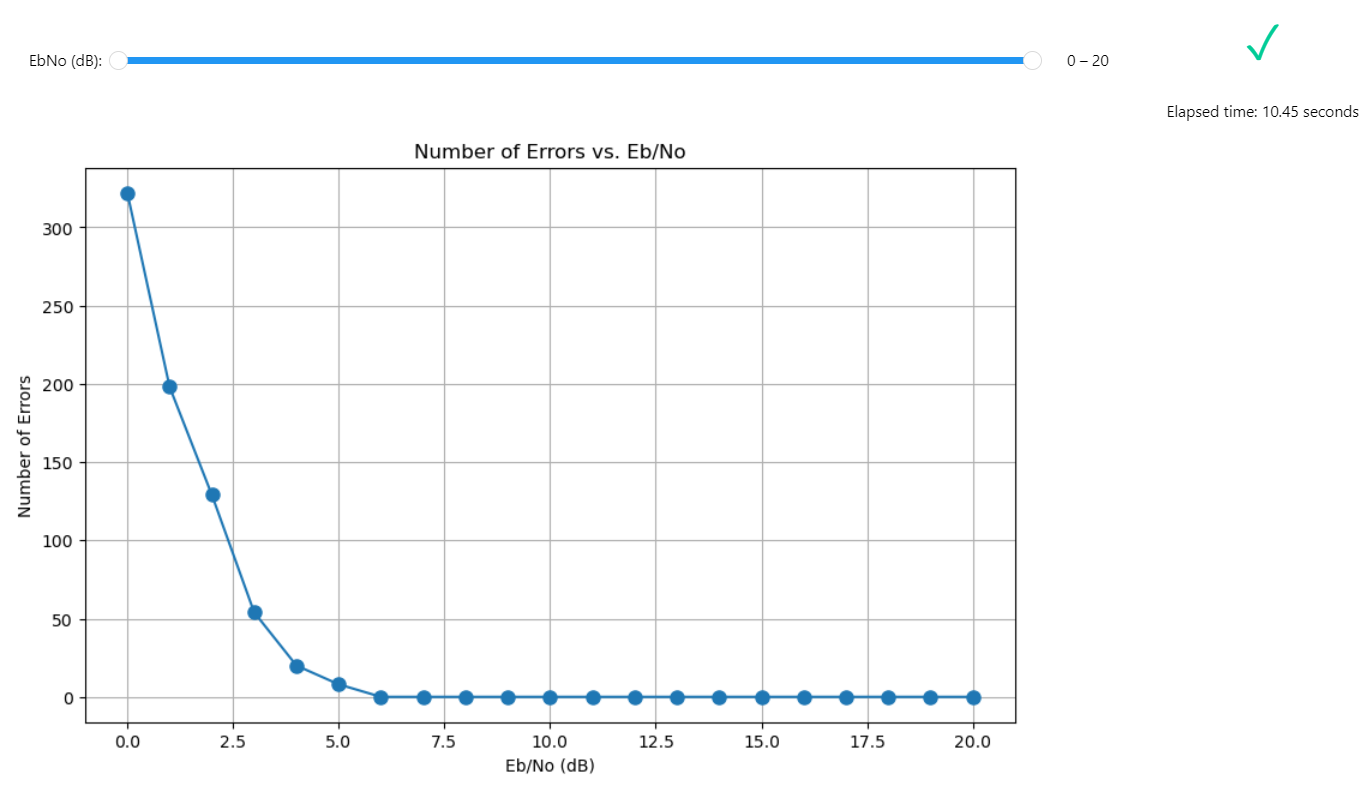
### 4.5.3 Αδυναμίες και Προκλήσεις

Δεν παρατηρούνται αξιοσημείωτες αδυναμίες στην προσομοίωση αυτή. Η μικρή απόκλιση σε πολύ χαμηλές τιμές Eb/N0 οφείλεται σε τυχαίο θόρυβο και είναι αναμενόμενη.

## 4.6 Lab Exercise 6: Διαμόρφωση FSK και MSK

### 4.6.1 Αποτελέσματα Προσομοίωσης

Τα αποτελέσματα που απεικονίζονται στα διαγράμματα υποδεικνύουν συνολικά μια επιτυχημένη προσέγγιση στην ανάλυση της απόδοσης διαφορετικών διαμορφώσεων και τεχνικών. Οι προσομοιώσεις καλύπτουν μια ευρεία γκάμα συστημάτων διαμόρφωσης (όπως MSK και FSK, τόσο συνεκτικό όσο και μη συνεκτικό), και οι μετρήσεις του ποσοστού σφαλμάτων bit (BER) και της φασματικής πυκνότητας ισχύος (PSD) είναι αρκετά κοντά στις θεωρητικές προσδοκίες.

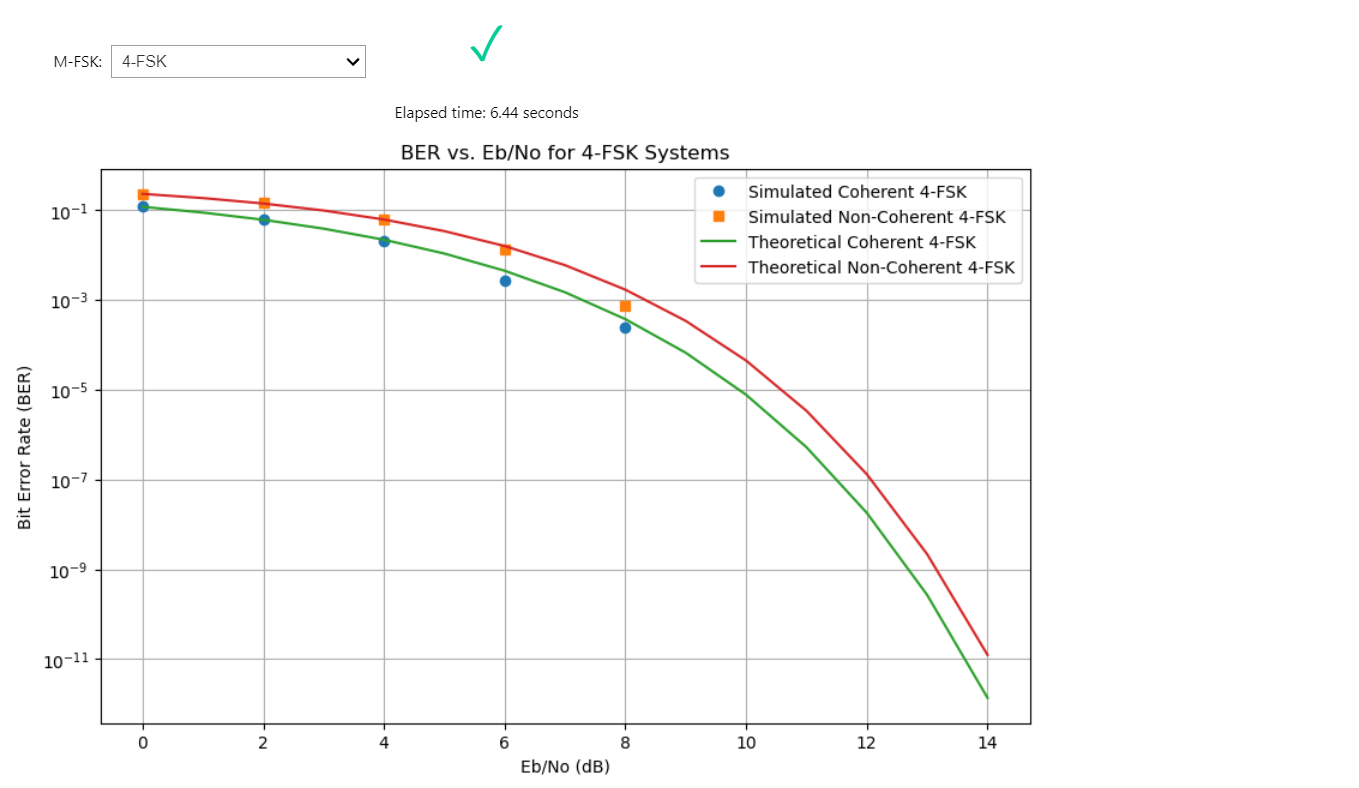


Σχήμα 49: Αριθμός λαθών

Οι καμπύλες BER στα περισσότερα διαγράμματα ακολουθούν την αναμενόμενη θεωρητική πορεία. Οι αποκλίσεις που παρατηρούνται σε ορισμένα διαστήματα, κυρίως στις χαμηλές τιμές Eb/N0, μπορούν να αποδοθούν σε στοχαστικά φαινόμενα, θόρυβο και την πιθανή επίδραση περιορισμένων δειγμάτων στις προσομοιώσεις. Παρά τις μικρές αυτές αποκλίσεις, η συνολική συμφωνία των προσομοιώσεων με τα θεωρητικά μοντέλα είναι εξαιρετική, ειδικά σε υψηλότερες τιμές Eb/N0 όπου η προσομοίωση παρουσιάζει σχεδόν ταυτοποίηση με τις θεωρητικές καμπύλες.

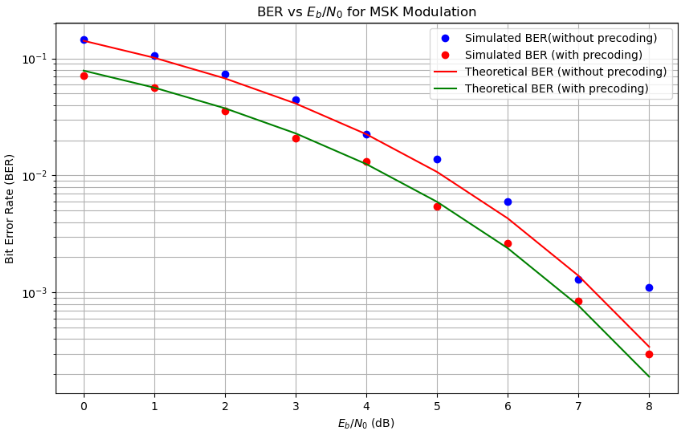
### 4.6.2 Ανάλυση Επιδόσεων

Ο χρόνος εκτέλεσης των προσομοιώσεων, όπως φαίνεται από τα στιγμιότυπα οθόνης, κυμαίνεται σε λογικά επίπεδα, επιτρέποντας την ολοκλήρωση των δοκιμών χωρίς μεγάλες καθυστερήσεις. Τα διαγράμματα που παρουσιάζουν PSD για τις συνεκτικές και μη συνεκτικές διαμορφώσεις FSK δείχνουν σαφείς διαφορές στη φασματική αποδοτικότητα, επιβεβαιώνοντας τη θεωρία. Τα αποτελέσματα αποτυπώνουν με ακρίβεια τη διαφορά στη συμπεριφορά των διαμορφώσεων αυτών, όπως αναμένεται από την ανάλυση των συχνοτήτων τους.



Σχήμα 50: BER M - FSK

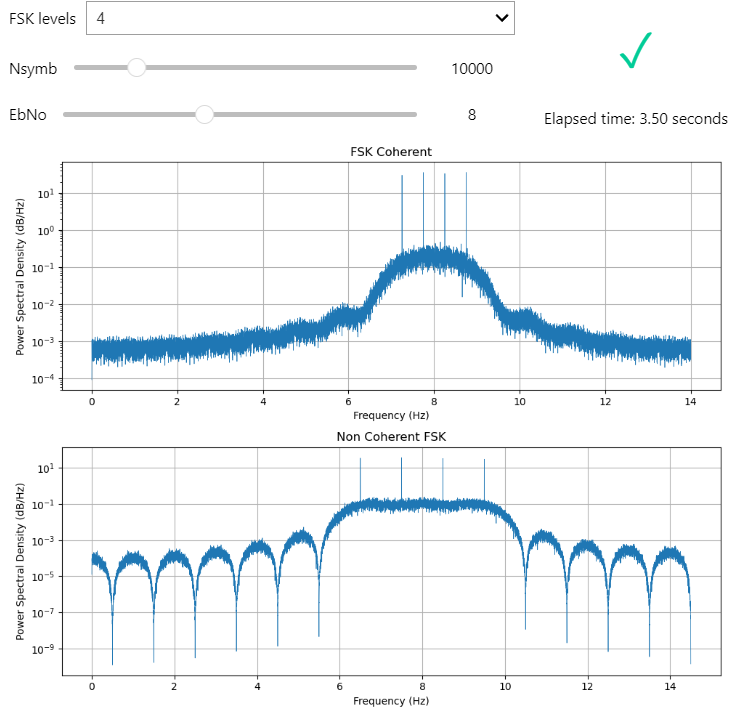
Συνολικά, τα διαγράμματα παρέχουν μια σαφή και ακριβή εικόνα της απόδοσης των διαφόρων συστημάτων διαμόρφωσης σε σχέση με το θόρυβο και τη φασματική απόδοση. Οι προσομοιώσεις, παρά τις μικρές αποκλίσεις, επιβεβαιώνουν τις θεωρητικές προσδοκίες και είναι επαρκώς αξιόπιστες. Ο χρόνος εκτέλεσης είναι γρήγορος και τα αποτελέσματα των προσομοιώσεων ενισχύουν τα συμπεράσματα σχετικά με την απόδοση αυτών των συστημάτων, προσφέροντας χρήσιμες πληροφορίες για τη βελτιστοποίηση των τεχνικών διαμόρφωσης.



Σχήμα 51: BER MSK

### 4.6.3 Αδυναμίες και Προκλήσεις

Παρά τη συνολική επιτυχία, θα μπορούσαν να υπάρξουν περαιτέρω βελτιώσεις στις προσομοιώσεις για τις χαμηλές τιμές Eb/N0, όπου παρατηρούνται μικρές αποκλίσεις. Αυτό ίσως οφείλεται είτε σε παράγοντες θορύβου είτε σε ατελή αντιστοίχιση του μοντέλου θορύβου στις προσομοιώσεις. Οι αδυναμίες αυτές, ωστόσο, δεν επηρεάζουν σημαντικά την εγκυρότητα των συμπερασμάτων. Επιπλέον, οι προσομοιώσεις παρουσιάζουν ικανοποιητική σταθερότητα, ακόμη και για πιο περίπλοκες διαμορφώσεις όπως το MSK με precoding.



Σχήμα 52: Φασματική πυκνότητα ισχύος

Κεφάλαιο 5

# Μελλοντικές επεκτάσεις

Ενσωμάτωση Προηγμένων Εννοιών

Η επέκταση της ιστοσελίδας με υλικό από το μάθημα "Ψηφιακές Επικοινωνίες II" αποτελεί φυσική εξέλιξη για την πλατφόρμα, καθώς προσφέρει στους φοιτητές πρόσβαση σε πιο προχωρημένες γνώσεις στον τομέα των τηλεπικοινωνιών. Το μάθημα αυτό καλύπτει θεμελιώδεις έννοιες όπως τα προχωρημένα συστήματα διαμόρφωσης (π.χ. QAM, OFDM), Γραμμικοί Τμηματικοί Κώδικες (Linear block codes), Κυκλικοί κώδικες, Κωδικοποίηση Reed-Solomon, Συνελικτική κωδικοποίηση. Η παροχή σημειώσεων και υλικού μέσω της πλατφόρμας μπορεί να προσφέρει στους φοιτητές μια διαδραστική και ολοκληρωμένη εκπαιδευτική εμπειρία, ενισχύοντας την κατανόηση αυτών των θεμάτων.

Ανάπτυξη Νέων Διαδραστικών Στοιχείων με προηγμένες διαδραστικές δυνατότητες

Η μελλοντική ανάπτυξη της πλατφόρμας μπορεί να περιλαμβάνει την ενσωμάτωση πιο εξελιγμένων διαδραστικών εργαλείων. Για παράδειγμα, η χρήση διαδραστικών διαγραμμάτων και τρισδιάστατων αναπαραστάσεων μπορεί να επιτρέψει στους φοιτητές να πειραματιστούν με διάφορες παραμέτρους συστημάτων επικοινωνίας, όπως η ισχύς σήματος, το εύρος ζώνης, και η διάδοση των σημάτων. Οι φοιτητές θα μπορούσαν να εκτελούν προσομοιώσεις, παρακολουθώντας σε πραγματικό χρόνο πώς επηρεάζονται τα συστήματα επικοινωνίας από τις αλλαγές σε παραμέτρους όπως το σήμα θορύβου και η απόσταση μετάδοσης.

Χρήση Βιβλιοθηκών Plotly και Bokeh

Για την υλοποίηση των διαδραστικών αυτών δυνατοτήτων, μπορούν να αξιοποιηθούν βιβλιοθήκες όπως οι **Plotly** και **Bokeh**, οι οποίες επιτρέπουν τη δημιουργία διαδραστικών γραφικών απεικονίσεων. Οι βιβλιοθήκες αυτές παρέχουν την ευελιξία για τη δημιουργία γραφημάτων που επιτρέπουν την άμεση αλληλεπίδραση των χρηστών με τα δεδομένα, γεγονός που μπορεί να προσφέρει στους φοιτητές μια πιο οπτική και πρακτική κατανόηση των τηλεπικοινωνιακών εννοιών.

Η προσθήκη σημειώσεων και ασκήσεων από το μάθημα "Ψηφιακές Επικοινωνίες II", σε συνδυασμό με την ανάπτυξη νέων διαδραστικών στοιχείων, αποτελεί σημαντική μελλοντική βελτίωση της πλατφόρμας. Η ενσωμάτωση αυτών των χαρακτηριστικών όχι μόνο θα ενισχύσει την εκπαιδευτική εμπειρία των φοιτητών, αλλά και θα τους επιτρέψει να εφαρμόσουν τις θεωρητικές τους γνώσεις σε πρακτικές και διαδραστικές εφαρμογές, προετοιμάζοντάς τους καλύτερα για την επαγγελματική τους πορεία στον τομέα των τηλεπικοινωνιών.

Κεφάλαιο 6

# Σύνοψη

## 6.1 Συνοπτική παρουσίαση του σκοπού

Ο σκοπός της διπλωματικής αυτής εργασίας είναι η ανάπτυξη διαδραστικών εργαστηριακών ασκήσεων για την εκπαίδευση στα βασικά θέματα της Ψηφιακής Επικοινωνίας, αξιοποιώντας την γλώσσα προγραμματισμού Python. Η ανάγκη για εκσυγχρονισμό των ασκήσεων και για μια περισσότερο διαδραστική προσέγγιση προκύπτει από την αυξανόμενη σημασία της πρακτικής εκπαίδευσης σε θέματα Ψηφιακής Επεξεργασίας Σήματος και διαμόρφωσης σήματος, ειδικά σε σύγχρονα συστήματα τηλεπικοινωνιών. Η Python, σε συνδυασμό με εργαλεία όπως το Jupyter Book, παρέχει τη δυνατότητα δημιουργίας ευέλικτων περιβαλλόντων για την ανάλυση σημάτων και την κατανόηση της επίδοσης των τηλεπικοινωνιακών συστημάτων σε πραγματικό χρόνο.

Οι κύριοι στόχοι της εργασίας περιλαμβάνουν:

* Την αντικατάσταση των εργαστηριακών ασκήσεων που αρχικά υλοποιήθηκαν σε MATLAB, με ασκήσεις σε Python για μεγαλύτερη διαδραστικότητα και προσαρμοστικότητα.
* Την ενσωμάτωση μεθόδων προσομοίωσης των διαμορφώσεων και των τεχνικών ψηφιακής επεξεργασίας σήματος που χρησιμοποιούνται στις σύγχρονες τηλεπικοινωνίες.
* Τη δημιουργία ενός εύχρηστου και διαισθητικού περιβάλλοντος μάθησης, το οποίο επιτρέπει στους φοιτητές να εφαρμόσουν τη θεωρία που διδάσκονται σε πρακτικές ασκήσεις.

## 6.2 Περίληψη κύριων βημάτων και μεθοδολογίας

Η μεθοδολογία που ακολουθήθηκε για την επίτευξη των στόχων της εργασίας περιλάμβανε τα εξής κύρια βήματα:

1. Ανάλυση και Μετατροπή Υφιστάμενων Ασκήσεων: Οι αρχικές εργαστηριακές ασκήσεις, οι οποίες ήταν γραμμένες σε MATLAB, μετετράπησαν σε Python. Αυτό προσέφερε μεγαλύτερη ευελιξία, καθώς η Python είναι μια γλώσσα προγραμματισμού που επιτρέπει τη δημιουργία πιο διαδραστικών εφαρμογών και διαθέτει ισχυρά εργαλεία για προσομοιώσεις σημάτων.
2. Δημιουργία Διαδραστικών Εφαρμογών με Jupyter Book: Η Python, σε συνδυασμό με το Jupyter Book, χρησιμοποιήθηκε για να δημιουργηθεί μια πλατφόρμα που επιτρέπει στους φοιτητές να αναπτύσσουν και να εκτελούν τις ασκήσεις. Το Jupyter Book παρέχει τη δυνατότητα ενσωμάτωσης διαδραστικών γραφικών στοιχείων, όπως ολισθητήρες (sliders) και γραφήματα σε πραγματικό χρόνο. Οι φοιτητές μπορούν να μεταβάλλουν παραμέτρους, όπως το Eb/N0 ή το πλήθος των συμβόλων, και να βλέπουν αμέσως τα αποτελέσματα σε μορφή γραφημάτων, γεγονός που ενισχύει την κατανόηση της θεωρίας.
3. Προσομοιώσεις Τηλεπικοινωνιακών Συστημάτων: Η εργασία επικεντρώνεται στις προσομοιώσεις διάφορων διαμορφώσεων ψηφιακού σήματος (L-ASK, QAM, PSK, FSK, MSK), καθώς και στη σηματοδότηση Nyquist και τα προσαρμοσμένα φίλτρα. Οι προσομοιώσεις αυτές πραγματοποιούνται με τη χρήση βιβλιοθηκών της Python, όπως η NumPy και η SciPy, οι οποίες παρέχουν ισχυρά εργαλεία για την ανάλυση σημάτων και την εκτέλεση πολύπλοκων μαθηματικών πράξεων.
4. Ενσωμάτωση Θεωρητικών Εννοιών και Σύγκριση με Πραγματικά Δεδομένα: Οι ασκήσεις σχεδιάστηκαν με σκοπό να παρέχουν στους φοιτητές μια ολοκληρωμένη εικόνα της θεωρίας και της πράξης. Τα προσομοιωμένα αποτελέσματα συγκρίνονται με τις θεωρητικές καμπύλες, επιτρέποντας στους φοιτητές να κατανοήσουν καλύτερα τη συμπεριφορά των συστημάτων επικοινωνίας σε διαφορετικές συνθήκες θορύβου ή ισχύος.

## 6.3 Βασικά Αποτελέσματα

Τα αποτελέσματα της εργασίας επιβεβαιώνουν την αποτελεσματικότητα της προσομοίωσης και της διαδραστικότητας στις ασκήσεις Ψηφιακών Επικοινωνιών.

**Ακρίβεια Προσομοιώσεων:** Τα προσομοιωμένα διαγράμματα BER (Bit Error Rate) που παράχθηκαν για τις διαμορφώσεις MSK και FSK παρουσιάζουν υψηλή συμφωνία με τις θεωρητικές καμπύλες, ιδιαίτερα για υψηλές τιμές του Eb/N0. Οι αποκλίσεις που παρατηρούνται σε χαμηλότερες τιμές Eb/N0 είναι μικρές και αναμενόμενες λόγω του θορύβου και των περιορισμών της προσομοίωσης.

**Ευέλικτη Προσαρμογή Παραμέτρων**: Η χρήση διαδραστικών οργάνων (όπως ολισθητήρες) επιτρέπει στους χρήστες να τροποποιούν παραμέτρους, όπως το πλήθος των συμβόλων (Nsymb), τα επίπεδα FSK, και το Eb/N0, και να παρατηρούν άμεσα την επίδραση αυτών των παραμέτρων στην απόδοση του συστήματος. Αυτό προσφέρει στους φοιτητές μια πιο άμεση κατανόηση των θεωρητικών εννοιών μέσω της πρακτικής εφαρμογής.

**Οπτικοποίηση και Ανάλυση Φασματικής Πυκνότητας:** Τα διαγράμματα φασματικής πυκνότητας ισχύος (PSD) που προκύπτουν από την προσομοίωση FSK αποδεικνύουν σαφείς διαφορές μεταξύ της συνεκτικής και μη συνεκτικής διαμόρφωσης, προσφέροντας μια πρακτική απεικόνιση των θεωρητικών εννοιών σχετικά με την απόδοση του συστήματος και τη φασματική αποδοτικότητα.

## 6.4 Συμπεράσματα

Η εργασία επιτυγχάνει τον στόχο της δημιουργίας μιας διαδραστικής πλατφόρμας εκπαίδευσης στις Ψηφιακές Επικοινωνίες. Μέσα από την αντικατάσταση των εργαστηριακών ασκήσεων από Python, παρέχεται ένα πιο ευέλικτο και προσαρμόσιμο περιβάλλον μάθησης, που επιτρέπει στους φοιτητές να αλληλεπιδρούν με τα δεδομένα σε πραγματικό χρόνο και να εξερευνούν τις επιδόσεις των συστημάτων τηλεπικοινωνιών.

Τα αποτελέσματα δείχνουν ότι οι προσομοιώσεις προσφέρουν ακριβή μοντέλα, τα οποία συμφωνούν με τις θεωρητικές αναλύσεις, ενώ η διαδραστική φύση της πλατφόρμας ενισχύει την κατανόηση των φοιτητών. Οι φοιτητές μπορούν να πειραματιστούν με διαφορετικά σενάρια, να παρακολουθήσουν την επίδραση του θορύβου και άλλων παραμέτρων στην απόδοση, και να εμβαθύνουν στην κατανόηση της θεωρίας μέσα από την πρακτική εφαρμογή της.

Ως μελλοντική προοπτική, προτείνεται η περαιτέρω ανάπτυξη της πλατφόρμας με περισσότερες εργαστηριακές ασκήσεις και η ενσωμάτωση περισσότερων σύγχρονων διαμορφώσεων και εργαλείων. Η ανάπτυξη της ψηφιακής εκπαίδευσης μέσω αυτών των εργαλείων έχει τη δυνατότητα να βελτιώσει την εκπαιδευτική εμπειρία και να ενισχύσει τις γνώσεις των φοιτητών στις ψηφιακές επικοινωνίες.

# Πηγές

[1] Ν. Μήτρου, *Ψηφιακές Επικοινωνίες - Συνοπτική Θεωρία και Εργαστήριο,*Εκδόσεις ΚΑΛΛΙΠΟΣ, 2016, (Ηλεκτρονικό, Ανοικτής Πρόσβασης, <https://hdl.handle.net/11419/6044>)

[2] Εργαστηριακή άσκηση 1 Εξοικείωση με το MATLAB - Σήματα Διακριτού χρόνου, <https://helios.ntua.gr/pluginfile.php/23213/mod_resource/content/10/DC-Lab1-2024.pdf>

[3] Εργαστηριακή Άσκηση 2 Σχεδιασμός/υλοποίηση ψηφιακών φίλτρων FIR με το MATLAB, <https://helios.ntua.gr/pluginfile.php/23215/mod_resource/content/12/DC-Lab2-2024.pdf>

[4] Εργαστηριακή Άσκηση 3 Προσαρμοσμένα φίλτρα και L-ASK, <https://helios.ntua.gr/pluginfile.php/23217/mod_resource/content/12/DC-Lab3-2023-24.pdf>

[5] Εργαστηριακή Άσκηση 4 Σηματοδοσία Nyquist, <https://helios.ntua.gr/pluginfile.php/23219/mod_resource/content/8/DC-Lab4-Nyquist_signaliing-ASK-2024.pdf>

[6] Εργαστηριακή Άσκηση 5 QAM-PSK, <https://helios.ntua.gr/pluginfile.php/23221/mod_resource/content/8/DC-Lab5-QAM-PSK.pdf>

[7] Εργαστηριακή Άσκηση 6 FSK-MSK, <https://helios.ntua.gr/pluginfile.php/23227/mod_resource/content/9/DC-Lab6-FSK_MSK.pdf>

[8] Επίσημη τεκμηρίωση του IPyWidgets, <https://ipywidgets.readthedocs.io/en/stable/>

[9] Widgets Gallery στην Επίσημη Τεκμηρίωση, <https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html>

[10] Python libraries, <https://docs.python.org/3/library/index.html>

[11] The Executable Book Project. (n.d.), <https://myst-parser.readthedocs.io/>

[12] The Executable Book Project. (n.d.). Jupyter Book,  [https://jupyterbook.org/](%20https://jupyterbook.org/)

[13] Georg Brandl, et al. (n.d.). Sphinx Documentation, <https://www.sphinx-doc.org/>

[14] The Executable Book Project. (n.d.). Thebe: Turn static documents into live code, <https://thebe.readthedocs.io/>

# Παράρτημα Ι

## Πλήρης Οδηγός Εγκατάστασης

Η δημιουργία και δημοσίευση ενός Jupyter Book είναι μια αποτελεσματική μέθοδος για τη συγγραφή και διανομή διαδραστικών βιβλίων που περιλαμβάνουν κείμενο και εκτελέσιμο κώδικα. Σε αυτόν τον οδηγό, περιγράφεται:

* [Η διαδικασία εγκατάστασης της Python](#_Εγκατάσταση_της_Python)
* [Των απαραίτητων βιβλιοθηκών Python με Conda](#_Εγκατάσταση_των_βιβλιοθηκών)
* [Η εγκατάσταση της βιβλιοθήκης Jupyter Book](#_Εγκατάσταση_Jupyter_Book)
* [Η δημιουργία ενός Jupyter Book](#_Δημιουργία_Jupyter_Book)
* [Η δημοσίευσή του μέσω GitHub.](#_Ανέβασμα_στο_GitHub)

**Εγκατάσταση της Python στον Υπολογιστή**

Αρχικά, απαιτείται η εγκατάσταση της Python. Ακολουθήστε τα παρακάτω βήματα:

1. Κατεβάστε το εκτελέσιμο αρχείο της Python από την επίσημη ιστοσελίδα: <https://www.python.org/downloads/>
2. Ακολουθήστε τον οδηγό εγκατάστασης και κατά την εγκατάσταση, βεβαιωθείται ότι έχετε επιλέξει την επιλογή "Add Python to PATH".
3. Ολοκληρώστε την εγκατάσταση ακολουθώντας τις οδηγίες.

Μετά την εγκατάσταση, ανοίξτε ένα τερματικό (Command Prompt ή PowerShell) και επιβεβαιώστε την επιτυχημένη εγκατάσταση με την εντολή:

**python –-version**

**Εγκατάσταση των βιβλιοθηκών python με Conda**

Η χρήση του Conda, ενός διαχειριστή πακέτων και περιβαλλόντων για Python, διευκολύνει την εγκατάσταση των απαραίτητων βιβλιοθηκών για την ανάπτυξη της εφαρμογής μέσω του Jupyter Book. Ακολουθήστε τις παρακάτω εντολές για να εγκαταστήσετε τις βιβλιοθήκες μέσω Conda:

1. Κατεβάστε το Conda από την επίσημη ιστοσελίδα:

<https://www.anaconda.com/download/success>

1. Ακολουθήστε τον οδηγό εγκατάστασης.
2. Δημιουργία περιβάλλοντος Conda:

**conda create --name myenv python=3.9**

**conda activate myenv**

1. Εγκατάσταση του **Jupyter Notebook**:

**conda install -c conda-forge notebook**

1. Εγκατάσταση βιβλιοθηκών για επιστημονικούς υπολογισμούς:

**conda install -c anaconda numpy**

**conda install -c anaconda scipy**

**conda install veeresht::scikit-commpy**

1. Εγκατάσταση βιβλιοθηκών για γραφήματα:

**conda install -c conda-forge matplotlib**

1. Εγκατάσταση του **IPython** & **IPyWidgets** για διαδραστικές εφαρμογές:

**conda install anaconda::ipython**

**conda install anaconda::ipywidgets**

1. Requests

**conda install anaconda::requests**

**Εγκατάσταση Jupyter Book**

Για τη δημιουργία του Jupyter Book, πρέπει πρώτα να εγκαταστήσουμε το πακέτο του:

**pip install -U jupyter-book**

**Δημιουργία Jupyter Book**

Η παρακάτω ενότητα περιλαμβάνει τις εντολές δημιουργίας, κατασκευής και προβολής ενός Jupyter Book.

1. Εντολή δημιουργίας νέου Jupyter Book:

**jupyter-book create mybook**

1. Εντολές κατασκευής και προβολής του Jupyter Book:
2. Εντολή κατασκευής του Jupyter Book:

**jupyter-book build mybook/**

1. Για την προβολή του βιβλίου τοπικά, μεταβείτε στον φάκελο *\_build/html* και ανοίξτε το αρχείο *index.html*.
2. Εναλλακτικά, μπορείτε να χρησιμοποιήσετε τον παρακάτω server για την προβολή του βιβλίου:

**python -m http.server --directory mybook/\_build/html**

**Ανέβασμα στο GitHub και Δημοσίευση στο Διαδίκτυο**

Για να ανεβάσεις το βιβλίο στο GitHub και να το δημοσιεύσεις, ακολούθησε τα παρακάτω βήματα:

1. Δημιουργία GitHub Repository:
2. Συνδέσου στον λογαριασμό σου στο GitHub και δημιούργησε ένα νέο repository.
3. Κατεβάστε το Git από την επίσημη ιστοσελίδα:

<https://git-scm.com/downloads>

1. Ακολουθήστε τον οδηγό εγκατάστασης.
2. Ακολούθως, στο τερματικό του τοπικού σου συστήματος, από κάποιο φάκελο της επιλογής σας:

**git init**

**git add .**

**git commit -m "Initial commit"**

**git remote add origin <URL\_του\_Repository>**

**git push -u origin master**

1. Ακολουθήστε τον επίσημο οδηγό δημιουργίας GitHub Pages:

<https://docs.github.com/en/pages/quickstart#creating-your-website>

1. Εντολή δημοσίευσης του Jupyter Book

**jupyter-book publish gh-pages mybook/ --repository <URL\_του\_Repository>**

Η εντολή αυτή θα δημιουργήσει ένα branch gh-pages και θα δημοσιεύσει το βιβλίο στο διαδίκτυο μέσω της υπηρεσίας GitHub Pages.

1. Διανομή του Βιβλίου:

Μετά τη δημοσίευση, το βιβλίο θα είναι διαθέσιμο στη διεύθυνση https://<όνομα\_χρήστη>.github.io/<repository\_name>.

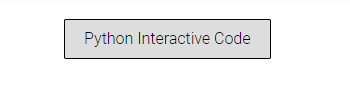
# Παράρτημα ΙΙ

## Οδηγός χρήσης του Jupyter Book με το Thebe

Αυτός ο οδηγός περιγράφει τη χρήση διαδραστικού κώδικα σε ένα Jupyter Book που χρησιμοποιεί το Thebe για εκτέλεση κώδικα. Ακολουθήστε τα παρακάτω βήματα για να εξασφαλίσετε ομαλή λειτουργία του διαδραστικού περιβάλλοντος.

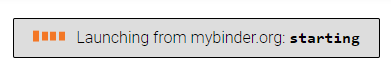
**Εκκίνηση Διαδραστικού Κώδικα**

* Αρχικά, πατήστε το κουμπί Interactive Code στην κορυφή της σελίδας.



*Σχήμα 53: Κουμπί Interactive code*

* Περιμένετε όσο η διαδικασία δείχνει μηνύματα όπως starting, fetching, launching, building κ.α.). Όταν το σύστημα εμφανίσει ότι είναι Ready, η σελίδα είναι έτοιμη για εκτέλεση διαδραστικού κώδικα.



Την πρώτη φορά θα χρειαστεί αναμονή 2’.



Σχήμα 54: Αναμονή για έτοιμο kernel

* Αν δείτε ότι η διαδικασία καθυστερεί για πάνω από 5 λεπτά χωρίς να προχωρεί στα επόμενα βήματα, ή αν κάνει fail, δοκιμάστε να κάνετε refresh (ανανέωση) στη σελίδα και επαναλάβετε τα παραπάνω βήματα.

**Επανεκκίνηση και Εκτέλεση Όλων**

* Αν θέλετε να επανεκκινήσετε όλο το σύστημα και να εκτελέσετε όλους τους κώδικες εκ νέου, πατήστε το κουμπί Restart and Run All.



Αυτή η λειτουργία θα τρέξει όλους τους κώδικες από την αρχή, καθαρίζοντας προηγούμενες εκτελέσεις.

Με αυτά τα βήματα, το Jupyter Book σας θα λειτουργεί διαδραστικά, επιτρέποντας την εκτέλεση και ανάλυση των εργαστηριακών ασκήσεων.

## Documentation: Basic Elements of a Jupyter Book

### Δομή του Jupyter Book

Ένα Jupyter Book αποτελείται από μια συλλογή αρχείων που οργανώνονται σε μια συγκεκριμένη δομή για να δημιουργηθεί το τελικό βιβλίο.

#### Βασικά Στοιχεία:

* **\_config.yml:** Το αρχείο ρυθμίσεων του Jupyter Book. Περιέχει παραμέτρους για το πώς θα δημιουργηθεί και θα εμφανιστεί το βιβλίο, όπως τον τίτλο, το λογότυπο, τις επεκτάσεις και άλλες παραμέτρους διαμόρφωσης.
* **\_toc.yml:** Το αρχείο Table of Contents (Πίνακας Περιεχομένων), το οποίο καθορίζει την ιεραρχία των κεφαλαίων και των ενοτήτων στο βιβλίο σας.
* **Αρχεία περιεχομένου:** Αυτά είναι τα αρχεία σε μορφή Markdown (.md) ή Jupyter Notebooks (.ipynb) που αποτελούν το περιεχόμενο του βιβλίου.

### Αρχεία Περιεχομένου

* **Markdown** (.md) (Markdown - Documentation, n.d.):

Αρχεία απλού κειμένου που χρησιμοποιούν συντακτική σήμανση για τη μορφοποίηση περιεχομένου, επιτρέποντας την εύκολη δημιουργία δομημένων εγγράφων με μορφές όπως τίτλους, λίστες, συνδέσμους και εικόνες.

* **MyST Markdown** (.md) (Myst Markdown - Documentation, n.d.):

Μια επέκταση του Markdown που προσθέτει επιπλέον λειτουργίες, όπως η υποστήριξη για μαθηματικά σύμβολα με LaTeX, διαδραστικά στοιχεία και τη δυνατότητα ενσωμάτωσης δομών κώδικα και παραπομπών, καθιστώντας το ιδανικό για τη δημιουργία τεχνικών και επιστημονικών εγγράφων.

* **Jupyter Notebooks** (.ipynb) (Jupyter Notebooks Files - Documentation, n.d.):

Τα Jupyter Notebooks μπορούν να περιλαμβάνουν κείμενο, κώδικα και αποτελέσματα, όλα ενσωματωμένα σε ένα έγγραφο. Αυτά τα αρχεία επιτρέπουν τη διαδραστικότητα, καθώς ο αναγνώστης μπορεί να εκτελεί τα μπλοκ κώδικα κατά την ανάγνωση του βιβλίου.

### Ρυθμίσεις του Jupyter Book

#### \_config.yml (Jupyter Book - Configuration File, n.d.):

Το **\_config.yml** καθορίζει τη γενική διαμόρφωση του βιβλίου. Παρακάτω παρατίθενται όλες οι ρυθμίσεις που μπορεί να περιλαμβάνει το αρχείο μαζί με μία σύντομη εξήγηση για κάθε μία:

####################################################################################

# A default configuration that will be loaded for all jupyter books

# Users are expected to override these values in their own `\_config.yml` file.

# This is also the "master list" of all allowed keys and values.

####################################################################################

# Book settings

# The title of the book. Will be placed in the left navbar.

title: My Jupyter Book

# The author of the book

author: The Jupyter Book community

# Copyright year to be placed in the footer

copyright: "2023"

# A path to the book logo

logo: ""

# Patterns to skip when building the book. Can be glob-style (e.g. "\*skip.ipynb")

exclude\_patterns: [\_build, Thumbs.db, .DS\_Store, "\*\*.ipynb\_checkpoints"]

# Auto-exclude files not in the table of contents (toc)

only\_build\_toc\_files: false

####################################################################################

# Execution settings

# Whether to execute notebooks at build time. Must be one of ("auto", "force", "cache", "off")

execute:

  execute\_notebooks: auto

# A path to the Jupyter cache that will be used to store execution artifacts

  cache: ""

# A list of patterns to skip in execution (e.g. notebooks that take too long to execute)

  exclude\_patterns: []

# The maximum time (in seconds) each notebook cell is allowed to run

  timeout: 30

# If true, then a temporary directory will be used as the working directory (cwd)

  run\_in\_temp: false

# If false, execution stops when a code cell raises an error; otherwise, all cells run

  allow\_errors: false

# Defines how to handle stderr output ('show', 'remove', 'remove-warn', 'warn', etc.)

  stderr\_output: show

####################################################################################

# Parse and render settings

# MyST extensions to enable (list of extensions)

parse:

  myst\_enable\_extensions:

    # Enables colon fences for better block delimitation

    - colon\_fence

    # Enables the use of LaTeX-style math expressions

    - dollarmath

    # Converts bare links into proper hyperlinks

    - linkify

    # Allows using text substitution in markdown

    - substitution

    # Adds checklists to the Markdown files

    - tasklist

# URI schemes that will be recognized as external URLs in Markdown links

  myst\_url\_schemes: [mailto, http, https]

# Allows display math ($$) within inline contexts

  myst\_dmath\_double\_inline: true

###################################################################################

# HTML-specific settings

# Path to a favicon image for the website

html:

  favicon: ""

# Whether to add an "edit this page" button to pages

  use\_edit\_page\_button: false

# Whether to add a repository link button

  use\_repository\_button: false

# Whether to add an "open an issue" button

  use\_issues\_button: false

# Continuous numbering across parts and chapters in the table of contents

  use\_multitoc\_numbering: true

# A custom message that will appear at the bottom of the footer

  extra\_footer: ""

# Whether to include the home page in the left Navigation Bar

  home\_page\_in\_navbar: true

# The base URL for the book, used for creating image previews and social links

  baseurl: ""

# Analytics settings for the website (Google Analytics, Plausible Analytics, etc.)

  analytics:

    plausible\_analytics\_domain: ""

    plausible\_analytics\_url: "https://plausible.io/js/script.js"

    google\_analytics\_id: ""

# Enable comments on the site (via Hypothesis or Utterances)

  comments:

    hypothesis: false

    utterances: false

# A banner announcement at the top of the site

  announcement: ""

####################################################################################

# LaTeX-specific settings

# The LaTeX engine to use for PDF builds ('pdflatex', 'xelatex', etc.)

latex:

  latex\_engine: pdflatex

# Whether to use the sphinx-jupyterbook-latex for PDF builds

  use\_jupyterbook\_latex: true

####################################################################################

# Launch button settings

# The interface for interactive links ("classic", "jupyterlab")

launch\_buttons:

  notebook\_interface: classic

# The URL of the BinderHub (for executing code in the cloud)

  binderhub\_url: ""

# The URL of the JupyterHub (for interactive computing)

  jupyterhub\_url: ""

# Whether to add a Thebe button to pages (for interactive code cells)

  thebe: false

# The URL of Google Colab for cloud execution

  colab\_url: ""

# The URL of Deepnote for interactive notebooks

  deepnote\_url: ""

####################################################################################

# Repository settings

# The URL to your book's repository

repository:

  url: https://github.com/executablebooks/jupyter-book

# The path to your book's folder, relative to the repository root

  path\_to\_book: ""

# The branch of the repository to use when creating links

  branch: master

####################################################################################

# Advanced and power-user settings

# Additional Sphinx extensions to load (in addition to defaults)

sphinx:

  extra\_extensions: []

# Local Sphinx extensions to load (defined by "name: path")

  local\_extensions: []

# Whether to overwrite or recursively update the Sphinx configuration

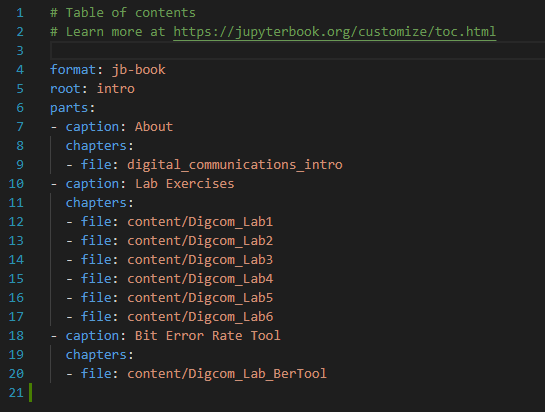
  recursive\_update: false

# Key-value pairs to directly override Sphinx configuration options

  config: {}

#### \_toc.yml (Jupyter Book - Table of Contents, n.d.):

Το αρχείο **\_toc.yml** οργανώνει το περιεχόμενο του βιβλίου σε κεφάλαια και υποκεφάλαια. Παρακάτω παρατίθεται η τελική δομή της εφαρμογής:



Σχήμα 59: \_toc.yml

Αυτό το αρχείο καθορίζει τη σειρά με την οποία τα αρχεία Markdown και Jupyter Notebooks θα εμφανιστούν στο βιβλίο.

### Διαδραστικότητα με το Thebe

Μια από τις πιο ισχυρές λειτουργίες του Jupyter Book είναι η χρήση του Thebe για την ενσωμάτωση διαδραστικού κώδικα απευθείας στην ιστοσελίδα. Με αυτόν τον τρόπο, οι χρήστες μπορούν να εκτελούν τα παραδείγματα κώδικα σε πραγματικό χρόνο χωρίς να χρειάζεται να κατεβάσουν τα αρχεία.

* **\_config.yml**

Για να ενεργοποιήσετε το Thebe, πρέπει να προσθέσετε τις παρακάτω ρυθμίσεις στο αρχείο \_config.yml:

**launch\_buttons:**

**thebe: true**

* **requirements.txt**

Αφού ενεργοποιηθεί η επιλογή `thebe: true` στο αρχείο “\_config.yml”, θα πρέπει να δημιουργηθεί ένας online kernel στο <https://mybinder.org/> . Για να επιτευχθεί αυτό, είναι απαραίτητο να υπάρχει στο πιο εξωτερικό φάκελο του GitHub repository το αρχείο “requirements.txt”, το οποίο θα περιλαμβάνει όλες τις βιβλιοθήκες που χρησιμοποιήθηκαν. Αυτό επιτρέπει στο Binder να εκτελέσει σωστά τον κώδικα μέσω του Thebe, διασφαλίζοντας ότι οι απαραίτητες εξαρτήσεις θα εγκατασταθούν αυτόματα.. Παρακάτω παρατίθεται το περιεχόμενο του “requirements.txt” της εφαρμογής:

numpy==1.24.3

scipy==1.10.1

matplotlib

ipython

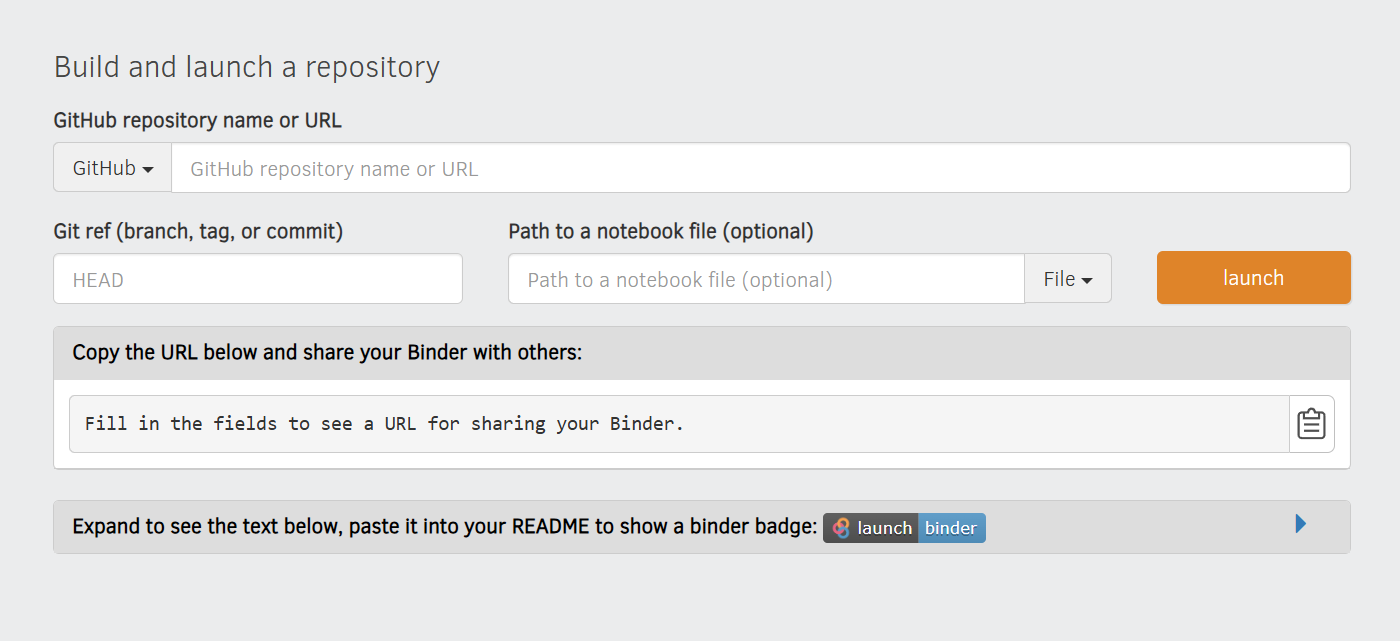
ipywidgets==7.7.2

requests

scikit-commpy

* **Binder**

Αφού συμπληρωθεί το αρχείο “**requirements.txt”,** μεταβαίνουμε στην ιστοσελίδα <https://mybinder.org/>. Εκεί, συμπληρώνουμε τα απαιτούμενα πεδία για το GitHub repository και πατάμε το κουμπί “**launch”**.:



Μετά από μια σύντομη αναμονή, το σύστημα θα δημιουργήσει τον online kernel, ο οποίος θα είναι έτοιμος για χρήση, επιτρέποντας την εκτέλεση του κώδικα μέσω του Thebe.

## Documentation of Lab Exercises

### Lab Exercise 1: Εισαγωγή στα Συστήματα Ψηφιακής Μετάδοσης

Δημιουργία Θορυβώδους Σήματος και   
Ανάλυση Φασματικής Πυκνότητας Ισχύος

# Loading animation HTML code

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create an HTML widget to display the loading animation

loader\_html2 = widgets.HTML(

  value=loading

)

# Create an HTML widget to display the elapsed time

timer\_html2 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# Output widget where plots will be displayed

plot\_output2 = widgets.Output()

# Function to calculate the next highest power of 2

def nextpow2(i):

    """

    Calculate the next power of 2 greater than or equal to the integer i.

    Parameters:

    i (int): Input integer.

    Returns:

    int: Next power of 2 greater than or equal to i.

    """

    n = 1

    while n < i:

        n \*= 2

    return n

# Function to update plots with input frequencies

def update\_plots2(freq1, freq2, freq3):

    """

    Update the plots based on the provided frequencies.

    Parameters:

    freq1 (int): Frequency of the first sine wave (Hz).

    freq2 (int): Frequency of the second sine wave (Hz).

    freq3 (int): Frequency of the third sine wave (Hz).

    """

    # Start timer and show loading animation

    loader\_html2.value = loading

    start\_time = time.time()

    # Sampling parameters

    Fs = 8000                    # Sampling frequency (Hz)

    Ts = 1/Fs                    # Sampling period (seconds)

    L = 8000                     # Length of signal (number of samples)

    T = L \* Ts                   # Total duration of signal (seconds)

    t = np.arange(0, L) \* Ts     # Time vector from 0 to T-Ts (inclusive)

    # Create the composite signal using the three input frequencies

    new\_x = np.sin(2\*np.pi\*freq1\*t) + 0.8\*np.sin(2\*np.pi\*freq2\*t) + np.sin(2\*np.pi\*freq3\*t)

    # Generate random noise

    rand\_n = np.random.randn(len(new\_x))  # Random noise with same length as new\_x

    # Check if any of the input frequencies are out of bounds

    if any(freq < 0 for freq in [freq1, freq2, freq3]) or any(freq > 3900 for freq in [freq1, freq2, freq3]):

        # If invalid frequency, clear the output and display a warning message

        with plot\_output2:

            plot\_output2.clear\_output(wait=True)

            warning\_html2 = widgets.HTML(

                value="<div style='color: black; background-color: #ffffcc; padding: 10px; border-radius: 5px; font-size: 16px; text-align: center;'>"

                      "<b>⚠️ Warning:</b> Frequency should not exceed 3900 and must be non-negative!</div>",

                placeholder='',

                description='',

            )

            display(warning\_html2)

            # Update the timer and loader to indicate completion

            elapsed\_time = time.time() - start\_time

            timer\_html2.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

            loader\_html2.value = done

    else:

        # If frequencies are valid, proceed to update the plots

        with plot\_output2:

            plot\_output2.clear\_output(wait=True)

            # Plotting setup: create a figure with 4 subplots

            fig, axs = plt.subplots(4, 1, figsize=(12, 20))

            # Time domain plot of random noise rand\_n

            axs[0].plot(t, rand\_n, color='#00CC96')

            axs[0].set\_title('Time domain plot of n')

            axs[0].set\_xlabel('t (sec)')

            axs[0].set\_ylabel('Amplitude')

            axs[0].grid(True)

            # Compute FFT parameters

            N = nextpow2(L)            # Length of Fourier transform (next power of 2)

            Fo = Fs / N                # Frequency resolution

            f = np.arange(0, N) \* Fo   # Frequency vector

            f\_shifted = f - Fs/2       # Shifted frequency vector for centered spectrum

            # Compute the FFT of the noise signal

            rand\_N = np.fft.fft(rand\_n, N)           # Compute DFT for N points

            rand\_N = np.fft.fftshift(rand\_N)         # Shift zero frequency component to center

            power\_n = np.multiply(rand\_N, np.conj(rand\_N)) / N / L  # Compute power spectral density

            # Frequency domain plot of the noise power spectrum

            axs[1].plot(f\_shifted, power\_n.real, color='#1F77B4')

            axs[1].set\_title('Frequency domain plot of n')

            axs[1].set\_xlabel('f (Hz)')

            axs[1].set\_ylabel('Amplitude')

            axs[1].grid(True)

            # Create the noisy signal by adding noise to the composite signal

            s = new\_x + rand\_n

            # Time domain plot of the noisy signal s

            axs[2].plot(t, s, color='#00CC96')

            axs[2].set\_title('Time domain plot of s')

            axs[2].set\_xlabel('t (sec)')

            axs[2].set\_ylabel('Amplitude')

            axs[2].grid(True)

            # Compute the FFT of the noisy signal s

            S = np.fft.fft(s, N)

            S = np.fft.fftshift(S)

            # Frequency domain plot of the noisy signal s

            axs[3].plot(f\_shifted, np.abs(S), color='#1F77B4')

            axs[3].set\_title('Frequency domain plot of s')

            axs[3].set\_xlabel('f (Hz)')

            axs[3].set\_ylabel('Magnitude')

            axs[3].grid(True)

            # Adjust layout to prevent overlap

            plt.tight\_layout()

            plt.show()

            # Update the timer and loader to indicate completion

            elapsed\_time = time.time() - start\_time

            timer\_html2.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

            loader\_html2.value = done

# Create three IntText widgets for input frequencies

freq1\_input2 = widgets.IntText(value=500, description='Freq 1 (Hz):', continuous\_update=False)

freq2\_input2 = widgets.IntText(value=1000, description='Freq 2 (Hz):', continuous\_update=False)

freq3\_input2 = widgets.IntText(value=1500, description='Freq 3 (Hz):', continuous\_update=False)

# Group the frequency inputs together in a vertical box

inputs\_box = widgets.VBox([freq1\_input2, freq2\_input2, freq3\_input2])

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox([loader\_html2, timer\_html2], layout=widgets.Layout(margin='0 0 0 40px'))

# Place the inputs and the loader/timer horizontally next to each other

ui = widgets.HBox([inputs\_box, loader\_timer\_box])

# Create an interactive output that updates plots when input frequencies change

out = widgets.interactive\_output(update\_plots2, {

    'freq1': freq1\_input2,

    'freq2': freq2\_input2,

    'freq3': freq3\_input2

})

# Display the UI and the plot output

display(ui, plot\_output2)

Πολλαπλασιασμός Σημάτων και Ανάλυση στο Πεδίο Χρόνου και Συχνότητας

# Loading animation HTML code

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create an HTML widget to display the loading animation

loader\_html3 = widgets.HTML(

  value=loading

)

# Create an HTML widget to display the elapsed time

timer\_html3 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# Output widget where plots or warnings will be displayed

plot\_output3 = widgets.Output()

# Function to calculate the next highest power of 2 greater than or equal to i

def nextpow2(i):

    """

    Calculate the next power of 2 greater than or equal to the given integer.

    Parameters:

    i (int): Input integer.

    Returns:

    int: Next power of 2 greater than or equal to i.

    """

    n = 1

    while n < i:

        n \*= 2

    return n

# Function to update plots based on the input frequencies

def update\_plots3(freq1, freq2, freq3):

    """

    Update the plots with the given input frequencies.

    Parameters:

    freq1 (int): Frequency of the first sine wave (Hz).

    freq2 (int): Frequency of the second sine wave (Hz).

    freq3 (int): Frequency of the third sine wave (Hz).

    """

    # Start the timer and show the loading animation

    loader\_html3.value = loading

    start\_time = time.time()

    # Check if any of the input frequencies are negative or exceed 500 Hz

    if any(freq < 0 for freq in [freq1, freq2, freq3]) or any(freq > 500 for freq in [freq1, freq2, freq3]):

        # If invalid frequencies, display a warning message

        with plot\_output3:

            plot\_output3.clear\_output(wait=True)

            warning\_html3 = widgets.HTML(

                value="<div style='color: black; background-color: #ffffcc; padding: 10px; border-radius: 5px; font-size: 16px; text-align: center;'>"

                      "<b>⚠️ Warning:</b> Frequency should not exceed 500 and must be non-negative!</div>",

                placeholder='',

                description='',

            )

            display(warning\_html3)

            # Update the elapsed time and loader to indicate completion

            elapsed\_time = time.time() - start\_time

            timer\_html3.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

            loader\_html3.value = done

    else:

        # Proceed with the plot if all frequencies are within the limit

        with plot\_output3:

            plot\_output3.clear\_output(wait=True)  # Clear any existing plots or warnings

            Fc = 1500  # Carrier frequency in Hz

            Fs = 8000  # Sampling frequency in Hz

            Ts = 1 / Fs  # Sampling period in seconds

            L = int(Fs)  # Length of signal (number of samples), for a 1-second signal

            t = np.linspace(0, 1, L, endpoint=False)  # Time vector from 0 to 1 second

            # Create the composite signal s as a sum of three sine waves

            s = np.sin(2 \* np.pi \* freq1 \* t) + 0.8 \* np.sin(2 \* np.pi \* freq2 \* t) + np.sin(2 \* np.pi \* freq3 \* t)

            # Create the carrier signal z at frequency Fc

            z = np.sin(2 \* np.pi \* Fc \* t)

            # Modulate the signal s using the carrier z (Amplitude Modulation)

            y = s \* z

            # Plotting the modulated signal in time domain

            fig, axs = plt.subplots(2, 1, figsize=(12, 10))

            # Plot the modulated signal y in the time domain

            axs[0].plot(t, y, color='#00CC96')

            axs[0].set\_title('Time domain plot of modulated signal y')

            axs[0].set\_xlabel('Time (sec)')

            axs[0].set\_ylabel('Amplitude')

            axs[0].set\_xlim(0, 0.2)  # Zoom into the first 0.2 seconds for clarity

            axs[0].set\_ylim(-2, 2)   # Set the y-axis limits

            axs[0].grid(True)

            # Fourier Transform to obtain the frequency domain representation

            N = nextpow2(L)  # Number of points for FFT, next power of 2 from L

            Y = np.fft.fft(y, N)  # Compute the FFT of the modulated signal

            Y = np.fft.fftshift(Y)  # Shift the zero frequency component to the center

            f = np.linspace(-Fs/2, Fs/2, N)  # Frequency vector for plotting

            # Plot the magnitude of the Fourier Transform

            axs[1].plot(f, np.abs(Y), color='#1F77B4')

            axs[1].set\_title('Frequency domain plot of modulated signal y')

            axs[1].set\_xlabel('Frequency (Hz)')

            axs[1].set\_ylabel('Magnitude')

            axs[1].grid(True)

            # Adjust layout and display the plots

            plt.tight\_layout()

            plt.show()

            # Update the elapsed time and loader to indicate completion

            elapsed\_time = time.time() - start\_time

            timer\_html3.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

            loader\_html3.value = done

# Create a label to show a warning above the text boxes

warning\_label = widgets.Label(value="Make sure the frequencies do not exceed 500 Hz")

# Create IntText widgets for frequency input, with default values and no continuous update

freq1\_input3 = widgets.IntText(value=100, description='Freq 1 (Hz):', continuous\_update=False)

freq2\_input3 = widgets.IntText(value=200, description='Freq 2 (Hz):', continuous\_update=False)

freq3\_input3 = widgets.IntText(value=300, description='Freq 3 (Hz):', continuous\_update=False)

# Group the frequency inputs and the warning label together in a vertical box

inputs\_box = widgets.VBox([freq1\_input3, freq2\_input3, freq3\_input3, warning\_label])

# Group the loader and timer together in a vertical box, with some left margin

loader\_timer\_box = widgets.VBox([loader\_html3, timer\_html3], layout=widgets.Layout(margin='0 0 0 40px'))

# Place the inputs and the loader/timer horizontally next to each other

ui = widgets.HBox([inputs\_box, loader\_timer\_box])

# Create an interactive output that updates when the frequency inputs change

widgets.interactive\_output(update\_plots3, {'freq1': freq1\_input3, 'freq2': freq2\_input3, 'freq3': freq3\_input3})

# Display the UI and the plot output widget

display(ui, plot\_output3)

Φασματική Ανάλυση Σήματος με Τμηματοποιημένη FFT και Σύγκριση με τη Μέθοδο Welch

# Loading animation HTML code

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create an HTML widget to display the loading animation

loader\_html4 = widgets.HTML(

  value=loading

)

# Create an HTML widget to display the elapsed time

timer\_html4 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# Function to calculate the next highest power of 2

def nextpow2(i):

    """

    Calculate the next power of 2 greater than or equal to the given integer.

    Parameters:

    i (int): Input integer.

    Returns:

    int: Next power of 2 greater than or equal to i.

    """

    n = 1

    while n < i:

        n \*= 2

    return n

# Custom implementation of the Welch's method for power spectral density estimation

def pwelch(x, Fs):

    """

    Estimate power spectral density using a custom implementation of Welch's method.

    Parameters:

    x (array\_like): Input signal.

    Fs (float): Sampling frequency.

    Returns:

    tuple: Frequencies and corresponding power spectral density estimates.

    """

    Ts = 1 / Fs                    # Sampling period

    L = np.size(x) + 1             # Length of the signal

    T = L \* Ts                     # Total duration

    N = 2^nextpow2(L)              # FFT length (next power of 2)

    Fo = Fs / N                    # Frequency resolution

    f = np.arange(0, N) \* Fo       # Frequency vector

    # Determine window size for segmenting the signal

    window\_size = nextpow2(np.size(x)/8)

    if (window\_size < 256):

        window\_size = 256

    windows = np.size(x) // (window\_size // 2) - 1  # Number of overlapping windows

    # Create an indexer for overlapping windows

    indexer = np.arange(window\_size)[None, :] + (window\_size // 2) \* np.arange(windows)[:, None]

    windowed\_x = x[indexer]  # Segment the signal into overlapping windows

    avg\_pwr = 0  # Initialize average power

    for window in windowed\_x:

        window = window \* np.hanning(np.size(window))  # Apply Hanning window

        L = np.size(window) + 1

        T = L \* Ts

        N = 2^nextpow2(L)              # FFT length (next power of 2)

        Fo = Fs / N

        f = np.arange(0, N) \* Fo

        window\_fft = np.fft.fft(window, N)  # Compute FFT

        power = np.multiply(window\_fft, np.conj(window\_fft)) / N / L  # Compute power spectral density

        avg\_pwr = avg\_pwr + power  # Accumulate power over all windows

    avg\_pwr = avg\_pwr / windows  # Average the power spectral density

    return f[np.arange(0, N // 2)], avg\_pwr[np.arange(0, N // 2)]  # Return one-sided spectrum

# Function to update plots based on slider value

def update\_plots3(Fs):

    """

    Update the plots based on the sampling frequency slider value.

    Parameters:

    Fs (int): Sampling frequency (Hz).

    """

    # Start timer and show loading animation

    loader\_html4.value = loading

    start\_time = time.time()

    L = 1000  # Length of the signal

    T = 1 / Fs  # Sampling period

    t1 = np.arange(0, L) \* T  # Time vector

    # Recompute signal x with new sampling frequency

    last\_x = np.sin(2 \* np.pi \* 30 \* t1) + 0.8 \* np.sin(2 \* np.pi \* 80 \* (t1 - 2)) + np.sin(2 \* np.pi \* 60 \* t1)

    # Compute power spectral density using custom pwelch function

    f1, Pxx1 = pwelch(last\_x, Fs)

    # Compute power spectral density using SciPy's signal.welch function

    f2, Pxx2 = signal.welch(last\_x, fs=Fs)

    # Create plots to compare the two methods

    fig, axs = plt.subplots(2, 1, figsize=(15, 10))

    # Plot custom pwelch result

    axs[0].plot(f1, Pxx1)

    axs[0].set(xlabel='Frequency (Hz)', ylabel='Power', title='Periodogram pwelch()')

    axs[0].grid()

    # Plot signal.welch result

    axs[1].plot(f2, Pxx2)

    axs[1].set(xlabel='Frequency (Hz)', ylabel='Power', title='Periodogram signal.welch()')

    axs[1].grid()

    # Adjust layout to prevent overlap

    plt.tight\_layout()

    # Show elapsed time

    elapsed\_time = time.time() - start\_time

    timer\_html4.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html4.value = done

# Create slider widget for sampling frequency Fs

Fs\_slider = widgets.IntSlider(

    value=500,

    min=100,

    max=2000,

    step=100,

    description='Sampling Frequency (Fs):',

    layout=Layout(width='auto', flex='1 1 auto'),

    style={'description\_width': 'initial'},

    continuous\_update=False

)

# Display the sliders and output

vbox\_layout = Layout(display='flex', flex\_flow='column', align\_items='center')

# Group the frequency inputs together in a horizontal box

inputs\_box = widgets.HBox([Fs\_slider], layout=Layout(flex='1 1 auto', width='auto'))

# Group the loader and timer together in a vertical box (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox([loader\_html4, timer\_html4], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Arrange the inputs and the loader/timer horizontally

ui = widgets.HBox([inputs\_box, loader\_timer\_box], layout=Layout(display='flex', justify\_content='center', width='100%', align\_items='center'))

# Create an interactive output that updates when the slider value changes

out = widgets.interactive\_output(update\_plots3, {'Fs': Fs\_slider})

# Display the UI and the output

display(ui, out)

### Lab Exercise 2: Ψηφιακά Φίλτρα

Τροποποίηση Κώδικα με Χρήση ifftshift/fftshift για Βελτιστοποίηση Φασματικής Ανάλυσης

# Loading animation HTML code for visual feedback during processing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html1 = widgets.HTML(

  value=loading

)

timer\_html1 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# Replace the URL with the URL of the raw content (sima.txt file)

url = "https://raw.githubusercontent.com/ntua-el17840/Interactive-Digital-Communications/main/test-book/\_static/sima.txt"

response = requests.get(url)

# Read the data from the text file, converting each line to a float, and store in array 's'

s = np.array([float(line) for line in response.text.splitlines()])

# Set the sampling frequency

Fs = 8192

# Compute the Power Spectral Density (PSD) of the signal 's' using Welch's method

f, Pxx = scipy.signal.welch(s, Fs)

# Number of frequency bins for FFT, set to the sampling frequency

N = Fs

# Create a frequency domain filter 'H' with ones in the passband and zeros in the stopband

H = np.concatenate((np.ones(N//8), np.zeros(N//4), np.ones(N//8)))

# Compute the impulse response 'h' by taking the inverse FFT of 'H' and keep the real part

h = np.fft.ifft(H, n=N).real

# Shift the impulse response to center it (align zero time to the center)

h = np.fft.fftshift(h)

# Extract different lengths of the impulse response for various filters

h32 = h[N//2-16:N//2+17]   # Filter with 33 coefficients

h64 = h[N//2-32:N//2+33]   # Filter with 65 coefficients

h128 = h[N//2-64:N//2+65]  # Filter with 129 coefficients

# Alternatively, ensure 'H' is correctly defined and calculated as before

# Create the frequency domain filter 'H' using horizontal stacking of arrays

H = np.hstack((np.ones(int(Fs/8)), np.zeros(int(Fs - Fs/4)), np.ones(int(Fs/8))))

# Compute the impulse response 'h' and take the real part

h = np.real(np.fft.ifft(H))

# Find the middle index of the impulse response

middle = int(len(h) / 2)

# Shift the impulse response to center it by rearranging the halves

h = np.hstack((h[middle:], h[:middle]))

# Extract various impulse responses of different lengths centered around the middle index

h32 = h[middle-16:middle+16]

h64 = h[middle-32:middle+32]

h128 = h[middle-64:middle+64]

h140 = h[middle-70:middle+70]

h256 = h[middle-128:middle+128]

# Create a dictionary to store different filter variants for easy access

h\_variants = {

    'h32': h[middle-16:middle+16],

    'h64': h[middle-32:middle+32],

    'h128': h[middle-64:middle+64],

    'h140': h[middle-70:middle+70],

    'h256': h[middle-128:middle+128],

}

# Create an output widget to display plots

output2 = widgets.Output()

output2.layout = Layout(width='auto', margin='0 auto')

# Function to generate and display the stem plot for the selected filter

def plot\_stem(h\_key):

    # Start the timer

    start\_time = time.time()

    with output2:

        loader\_html1.value = loading  # Show loading animation

        clear\_output(wait=True)       # Clear any previous output in the widget

        h\_data = h\_variants[h\_key]    # Get the impulse response data for the selected filter

        x\_values = np.arange(len(h\_data))  # Create an array of indices for plotting

        plt.close('all')              # Close any existing figures to prevent overlaps

        fig, ax = plt.subplots()      # Create a new figure and axes

        # Generate a stem plot of the impulse response

        markerline, stemlines, baseline = ax.stem(x\_values, h\_data, '-.')

        # Customize the baseline appearance

        plt.setp(baseline, 'color', 'k', 'linewidth', 2)

        # Set plot titles and labels

        plt.title('Stem plot of selected filter')

        plt.xlabel('Index')

        plt.ylabel('Amplitude')

        # Update the elapsed time and loader to indicate completion

        elapsed\_time = time.time() - start\_time

        timer\_html1.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html1.value = done

        # Display the plot

        plt.show()

# HTML Label for the visualization section

html\_label = widgets.HTML(

    value="<h2 style='font-weight: bold; font-size: 30px; text-align: center;'>Filter Visualization</h2>"

)

# Setup the dropdown widget for selecting different filters

dropdown = widgets.Dropdown(

    options=list(h\_variants.keys()),  # Options are the keys from h\_variants dictionary

    value='h32',                      # Default selected value

    description='Filter:'

)

# Function that updates the plot when the dropdown selection changes

def update\_plot(change):

    plot\_stem(change['new'])  # Call plot\_stem with the new selection

# Observe the dropdown for changes and call update\_plot when it changes

dropdown.observe(update\_plot, names='value')

# Layout settings for aligning items vertically and horizontally

vbox\_layout = Layout(

    display='flex',

    flex\_flow='column',

    align\_items='center',

    justify\_content='space-between'

)

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox(

    [loader\_html1, timer\_html1],

    layout=widgets.Layout(margin='0 0 0 40px')

)

# Arrange the dropdown and loader/timer horizontally

ui = widgets.HBox(

    [dropdown, loader\_timer\_box],

    layout=Layout(align\_items='center', justify\_content='center')

)

# Group the HTML label, UI components, and output widget in a vertical box layout

vbox\_final = widgets.VBox([html\_label, ui, output2])

# Display the final VBox containing all components

display(vbox\_final)

# Initial call to display the plot using the default value from the dropdown

plot\_stem(dropdown.value)

Σχεδιασμός Βαθυπερατού Φίλτρου με Ορθογωνικό και Hamming Παράθυρο

# Loading animation HTML code for visual feedback during processing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html2 = widgets.HTML(

  value=loading

)

timer\_html2 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# Compute frequency responses for various filters

freq32, resp32 = signal.freqz(h32)

freq64, resp64 = signal.freqz(h64)

freq128, resp128 = signal.freqz(h128)

freq140, resp160 = signal.freqz(h140)

freq256, resp256 = signal.freqz(h256)

# Create a dictionary to map filter names to their corresponding variables

filter\_dict = {

    'h32': h32,

    'h64': h64,

    'h128': h128,

    'h140': h140,

    'h256': h256

}

# Function to apply window and compute frequency responses

def compute\_filtered\_freq\_responses(window\_type='Rectangular'):

    """

    Compute the frequency responses of filters with the specified window type.

    Parameters:

    window\_type (str): Type of window to apply ('Rectangular', 'Hamming', or 'Kaiser').

    Returns:

    freqs (dict): Dictionary of frequency arrays for each filter.

    resps (dict): Dictionary of frequency responses for each filter.

    """

    freqs, resps = {}, {}

    for filt\_size in [32, 64, 128, 140, 256]:

        filt\_name = f'h{filt\_size}'  # Construct filter name as a string

        filt = filter\_dict[filt\_name]  # Access the filter array from the dictionary

        # Apply the specified window to the filter coefficients

        if window\_type == 'Hamming':

            filt = filt \* np.hamming(filt\_size)

        elif window\_type == 'Kaiser':

            beta = 14  # Beta value for Kaiser window, adjust as needed

            filt = filt \* np.kaiser(filt\_size, beta)

        # Compute the frequency response of the windowed filter

        freqs[filt\_name], resps[filt\_name] = signal.freqz(filt)

    return freqs, resps

# Initial computation with Rectangular (no window)

freqs, resps = compute\_filtered\_freq\_responses()

# Create an output widget to display plots

output3 = widgets.Output()

output3.layout = Layout(width='auto', margin='0 auto')

def update\_plot1(change=None):

    """

    Update the plot based on the selected window type and filters.

    Parameters:

    change: Optional parameter for observe callbacks (not used here).

    """

    # Start timer

    start\_time = time.time()

    # Get the selected window type from the dropdown

    window\_type = window\_type\_dropdown.value

    # Compute frequency responses with the selected window

    freqs, resps = compute\_filtered\_freq\_responses(window\_type)

    with output3:

        loader\_html2.value = loading  # Display loading animation

        clear\_output(wait=True)       # Clear previous output

        plt.figure(figsize=(10, 5))

        # Loop through each filter and plot if the corresponding checkbox is checked

        for filt in ['h32', 'h64', 'h128', 'h140', 'h256']:

            if checkboxes1[filt].value:

                w, h = freqs[filt], resps[filt]

                plt.plot(0.5 \* Fs \* w / np.pi, 20 \* np.log10(np.abs(h)), label=filt)

        plt.title(f'Frequency Response with {window\_type} Window')

        plt.xlabel('Frequency (Hz)')

        plt.ylabel('Magnitude (dB)')

        plt.xscale('linear')

        plt.grid(True)

        plt.legend()

        # Show elapsed time

        elapsed\_time = time.time() - start\_time

        timer\_html2.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html2.value = done  # Display completion checkmark

        plt.show()

# Create checkboxes for each filter

checkboxes1 = {f'h{size}': widgets.Checkbox(value=True, description=f'h{size}') for size in [32, 64, 128, 140, 256]}

# Observe changes in the checkboxes to update the plot

for cb in checkboxes1.values():

    cb.observe(update\_plot1, names='value')

# Dropdown for selecting the window type

window\_type\_dropdown = widgets.Dropdown(

    options=['Rectangular', 'Hamming', 'Kaiser'],

    value='Rectangular',

    description='Window Type:',

    style={'description\_width': 'initial'}

)

# Observe changes in the dropdown to update the plot

window\_type\_dropdown.observe(update\_plot1, names='value')

# HTML Label for the visualization

html\_label = widgets.HTML(

    value="<h2 style='font-weight: bold; font-size: 30px; text-align: center;'>Filter Frequency Response Visualization</h2>"

)

# VBox layout to align items

vbox\_layout = Layout(align\_items='center', justify\_content='center')

vbox\_checkboxes = widgets.VBox(list(checkboxes1.values()) + [window\_type\_dropdown], layout=vbox\_layout)

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox(

    [loader\_html2, timer\_html2],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Combine the checkboxes and loader/timer in a horizontal box

vbox\_loader = widgets.HBox([vbox\_checkboxes, loader\_timer\_box])

# Combine everything into a final VBox with the layout

vbox\_final = widgets.VBox([html\_label, vbox\_loader, output3], layout=vbox\_layout)

# Display the final layout

display(vbox\_final)

# Initial plot update to show default visualization

update\_plot1()

Σχεδιασμός Φίλτρου Parks-McClellan και Σύγκριση Αποκρίσεων Συχνότητας για Διάφορα Μήκη

# Loading animation HTML code for visual feedback during processing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html3 = widgets.HTML(

  value=loading

)

timer\_html3 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# Define a dictionary 'filters' containing equiripple filters of various lengths

filters = {

    'Equiripple Filter 32+1': scipy.signal.remez(33, [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs], [1, 0], Hz=Fs),

    'Equiripple Filter 64+1': scipy.signal.remez(65, [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs], [1, 0], Hz=Fs),

    'Equiripple Filter 128+1': scipy.signal.remez(129, [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs], [1, 0], Hz=Fs),

    'Equiripple Filter 140+1': scipy.signal.remez(141, [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs], [1, 0], Hz=Fs)

}

# Create an output widget to display the plot

output\_plot2 = widgets.Output()

output\_plot2.layout = Layout(width='auto', margin='0 auto')

# Function to update the plot based on selected filters

def update\_plot2(change=None):

    """

    Update the frequency response plot based on the selected filters.

    Parameters:

    change (dict): Dictionary containing information about the change event.

    """

    # Start timer

    start\_time = time.time()

    # Get the list of selected filters from the checkboxes

    selected\_filters = [cb.description for cb in checkboxes2 if cb.value]

    with output\_plot2:

        loader\_html3.value = loading  # Display loading animation

        clear\_output(wait=True)       # Clear previous output

        plt.figure(figsize=(10, 5))

        # Loop through each selected filter and plot its frequency response

        for filter\_name in selected\_filters:

            filter\_coeffs = filters[filter\_name]  # Get the filter coefficients

            # Compute the frequency response

            w, h = signal.freqz(filter\_coeffs, worN=8000)

            # Plot the magnitude of the frequency response on a semilogarithmic scale

            plt.semilogy(w \* Fs / (2 \* np.pi), np.abs(h), label=filter\_name)

        plt.title('Frequency Response')

        plt.xlabel('Frequency (Hz)')

        plt.ylabel('Gain')

        plt.legend()

        plt.grid(True)

        # Show elapsed time

        elapsed\_time = time.time() - start\_time

        timer\_html3.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html3.value = done  # Display completion checkmark

        plt.show()

# Create checkboxes for each filter to allow user selection

checkboxes2 = [widgets.Checkbox(value=True, description=name) for name in filters.keys()]

# Observe changes in the checkboxes to update the plot when selections change

for cb in checkboxes2:

    cb.observe(update\_plot2, names='value')

# HTML Label for the visualization

html\_label = widgets.HTML(

    value="<h2 style='font-weight: bold; font-size: 30px; text-align: center;'>Equiripple Filter Frequency Response</h2>"

)

# VBox layout to align items vertically and center them

vbox\_layout = Layout(align\_items='center', justify\_content='center')

# Create a vertical box containing the label and checkboxes

vbox\_checkboxes = widgets.VBox(

    [widgets.Label('Select Equiripple Filter Length:')] + checkboxes2,

    layout=vbox\_layout

)

# Group the loader and timer widgets vertically

loader\_timer\_box = widgets.VBox(

    [loader\_html3, timer\_html3],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Arrange the checkboxes and loader/timer horizontally

vbox\_loader = widgets.HBox([vbox\_checkboxes, loader\_timer\_box])

# Combine the HTML label, controls, and output plot into a final vertical box

vbox\_final = widgets.VBox([html\_label, vbox\_loader, output\_plot2], layout=vbox\_layout)

# Display the final layout

display(vbox\_final)

# Initial plot update to show default visualization

update\_plot2()

Σύγκριση Αποκρίσεων Συχνότητας Φίλτρου Parks-McClellan με Διαφορετικές Οριακές Συχνότητες

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Green \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }  /\* Starting point of the animation \*/

        100% { transform: rotate(360deg); }  /\* Ending point after one full rotation \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

    </div>

    """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html4 = widgets.HTML(

    value=loading  # Initially display the loading animation

)

timer\_html4 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for elapsed time

)

# Define a dictionary 'filters3' containing equiripple filters of various lengths

# Each filter is designed using the Parks-McClellan algorithm (remez function)

filters3 = {

    'Equiripple Filter 32+1': scipy.signal.remez(

        33,                        # Filter length (number of taps)

        [0, 0.11 \* Fs, 0.12 \* Fs, 0.5 \* Fs],  # Band edges in Hz

        [1, 0],                    # Desired amplitudes in each band (1 in passband, 0 in stopband)

        Hz=Fs                      # Sampling frequency

    ),

    'Equiripple Filter 64+1': scipy.signal.remez(

        65,

        [0, 0.11 \* Fs, 0.12 \* Fs, 0.5 \* Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 128+1': scipy.signal.remez(

        129,

        [0, 0.11 \* Fs, 0.12 \* Fs, 0.5 \* Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 140+1': scipy.signal.remez(

        141,

        [0, 0.11 \* Fs, 0.12 \* Fs, 0.5 \* Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 256+1': scipy.signal.remez(

        257,

        [0, 0.11 \* Fs, 0.12 \* Fs, 0.5 \* Fs],

        [1, 0],

        Hz=Fs

    )

}

# Create an output widget to display the plot

output\_plot3 = widgets.Output()

# Function to update the plot based on selected filters

def update\_plot3(change=None):

    """

    Update the frequency response plot based on the selected filters.

    Parameters:

    change (dict): Dictionary containing information about the change event.

    """

    # Start the timer and show the loading animation

    start\_time = time.time()

    selected\_filters = [cb.description for cb in checkboxes3 if cb.value]  # Get names of selected filters

    with output\_plot3:

        loader\_html4.value = loading  # Display the loading animation

        clear\_output(wait=True)       # Clear previous outputs

        plt.figure(figsize=(10, 5))   # Create a new figure for plotting

        # Loop through each selected filter and plot its frequency response

        for filter\_name in selected\_filters:

            filter\_coeffs = filters3[filter\_name]  # Retrieve filter coefficients from the dictionary

            # Compute the frequency response of the filter

            w, h = signal.freqz(filter\_coeffs, worN=8000)

            # Plot the magnitude response (gain) in decibels

            plt.semilogy(w \* Fs / (2 \* np.pi), np.abs(h), label=filter\_name)

        plt.title('Frequency Response')       # Set the plot title

        plt.xlabel('Frequency (Hz)')          # Label the x-axis

        plt.ylabel('Gain')                    # Label the y-axis

        plt.legend()                          # Display a legend

        plt.grid(True)                        # Add grid lines to the plot

        # Update the elapsed time and loader to indicate completion

        elapsed\_time = time.time() - start\_time                 # Calculate elapsed time

        timer\_html4.value = f"Elapsed time: {elapsed\_time:.2f} seconds"  # Update timer display

        loader\_html4.value = done                               # Display the completion checkmark

        plt.show()  # Display the plot

# Create checkboxes for each filter to allow user selection

checkboxes3 = [widgets.Checkbox(value=True, description=name) for name in filters3.keys()]

# Set up observers to call 'update\_plot3' when the value of a checkbox changes

for cb in checkboxes3:

    cb.observe(update\_plot3, names='value')

# HTML Label for the visualization section

html\_label3 = widgets.HTML(

    value="<h2 style='font-weight: bold; font-size: 30px; text-align: center;'>Equiripple Filter Frequency Response</h2>"

)

# VBox layout to align items vertically and center them

vbox\_layout3 = Layout(

    display='flex',

    flex\_flow='column',        # Arrange items in a column (vertical layout)

    align\_items='center',      # Center items horizontally

    justify\_content='space-between'  # Distribute space evenly

)

# Create a vertical box containing the label and checkboxes

vbox\_checkboxes3 = widgets.VBox(

    [widgets.Label('Select Equiripple Filter Length:')] + checkboxes3,  # Add a label and the checkboxes

    layout=vbox\_layout3

)

# Group the loader and timer widgets vertically

loader\_timer\_box = widgets.VBox(

    [loader\_html4, timer\_html4],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')  # Add left margin and auto width

)

# Arrange the checkboxes and loader/timer horizontally

vbox\_loader = widgets.HBox(

    [vbox\_checkboxes3, loader\_timer\_box]

)

# Combine the HTML label, controls, and output plot into a final vertical box

vbox\_final3 = widgets.VBox(

    [html\_label3, vbox\_loader, output\_plot3],  # Add the label, controls, and output to the VBox

    layout=vbox\_layout3

)

# Display the final layout in the notebook

display(vbox\_final3)

# Initial plot update to show default visualization

update\_plot3()

Φασματική Ανάλυση και Φιλτράρισμα Σήματος με Φίλτρα Parks-McClellan για Διαφορετικές Οριακές Συχνότητες

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }  /\* Starting point of the rotation \*/

        100% { transform: rotate(360deg); }  /\* Ending point after one full rotation \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

# These widgets will be updated during processing to provide user feedback

loader\_html6 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html6 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for elapsed time

)

# Sampling frequency

Fs = 8192  # Sampling frequency in Hz

# Signal duration

t = np.arange(0, 1.0, 1/Fs)  # Time vector from 0 to 1 second with step size 1/Fs

# Re-define the signal with new frequencies

# The signal 's\_new' is a sum of four sinusoids at different frequencies

s\_new = np.sin(2\*np.pi\*400\*t) + np.sin(2\*np.pi\*950\*t) + np.sin(2\*np.pi\*1500\*t) + np.sin(2\*np.pi\*3000\*t)

# Re-define the filters with different lengths

# We define a dictionary 'filters5' containing different equiripple filters

# Each filter is designed using the Parks-McClellan algorithm (remez function)

filters5 = {

    'Equiripple Filter 32+1 Q3': signal.remez(

        33,                           # Filter length (number of taps)

        [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs], # Band edges in Hz for Question 3

        [1, 0],                       # Desired amplitudes in each band (1 in passband, 0 in stopband)

        Hz=Fs                         # Sampling frequency

    ),

    'Equiripple Filter 32+1 Q4': signal.remez(

        33,

        [0, 0.11\*Fs, 0.12\*Fs, 0.5\*Fs], # Band edges in Hz for Question 4

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 64+1 Q3': signal.remez(

        65,

        [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 64+1 Q4': signal.remez(

        65,

        [0, 0.11\*Fs, 0.12\*Fs, 0.5\*Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 128+1 Q3': signal.remez(

        129,

        [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 128+1 Q4': signal.remez(

        129,

        [0, 0.11\*Fs, 0.12\*Fs, 0.5\*Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 140+1 Q3': signal.remez(

        141,

        [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs],

        [1, 0],

        Hz=Fs

    ),

    'Equiripple Filter 140+1 Q4': signal.remez(

        141,

        [0, 0.11\*Fs, 0.12\*Fs, 0.5\*Fs],

        [1, 0],

        Hz=Fs

    ),

}

# Create checkboxes for each filter

# Users can select which filters to include in the plot

checkboxes5 = [widgets.Checkbox(value=True, description=name) for name in filters5.keys()]

# Create an output widget to display the plot

output\_plot5 = widgets.Output()

# Function to update the plot based on selected filters

def update\_plot5(dummy=None):

    with output\_plot5:

        # Start timer and show the loading animation

        loader\_html6.value = loading

        start\_time = time.time()

        # Clear the current plot output

        output\_plot5.clear\_output(wait=True)

        # Start a new plot

        plt.figure(figsize=(14, 4))

        # Add original signal PSD (Power Spectral Density)

        f, Pxx\_den\_original = signal.welch(s\_new, Fs, nperseg=1024)

        plt.semilogy(f, Pxx\_den\_original, label='Original Signal')

        # Plot PSD for each selected filter

        for cb in checkboxes5:

            if cb.value:  # Only plot if the checkbox is checked

                filter\_name = cb.description

                filter\_coeffs = filters5[filter\_name]

                # Apply the filter to the signal

                filtered\_signal = signal.lfilter(filter\_coeffs, 1.0, s\_new)

                # Compute the PSD of the filtered signal

                f, Pxx\_den\_filtered = signal.welch(filtered\_signal, Fs, nperseg=1024)

                plt.semilogy(f, Pxx\_den\_filtered, label=filter\_name)

        # Formatting the plot

        plt.title('Power Spectral Density')

        plt.xlabel('Frequency (Hz)')

        plt.ylabel('Power/Frequency (dB/Hz)')

        plt.legend()

        plt.grid(which='both', axis='both')

        # Update the elapsed time and loader to indicate completion

        elapsed\_time = time.time() - start\_time

        timer\_html6.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html6.value = done

        plt.show()

# Attach the update\_plot function to the checkboxes

# Whenever a checkbox value changes, the plot will update

for cb in checkboxes5:

    cb.observe(update\_plot5, names='value')

# Initial call to update the plot with the default selections

update\_plot5()

# HTML Label for enhanced UI

html\_label = widgets.HTML(

    value="""

    <h2 style='font-weight: bold; font-size: 30px; text-align: center; margin: 20px 0 0 0;'>Power Spectral Density</h2>

    <h4 style='font-weight: bold; font-size: 16px; text-align: center; margin: 0 0 20px 0;'>(using Parks-McClellan filters)</h4>

    """

)

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox(

    [loader\_html6, timer\_html6],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Create a vertical box to hold the checkboxes

vbox\_checkboxes5 = widgets.VBox(checkboxes5)

# Arrange the checkboxes and loader/timer horizontally

ui = widgets.HBox(

    [vbox\_checkboxes5, loader\_timer\_box],

    layout=Layout(justify\_content='center', align\_items='center')

)

# Combine the HTML label, controls, and output plot into a final vertical box

out = widgets.VBox([html\_label, ui, output\_plot5])

# Display the checkboxes and the plot

display(out)

Σχεδιασμός Ζωνοπερατού Φίλτρου και Εφαρμογή του σε Σήμα μέσω Δύο Μεθόδων

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Starting position \*/

        100% { transform: rotate(360deg); } /\* Full rotation \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html7 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html7 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

# Define your input widgets for the frequencies f1 and f2

f1\_input = widgets.IntText(

    value=600,                      # Default value for f1 in Hz

    description='f1 (Hz):',         # Label for the widget

    continuous\_update=False         # Update only when the user finishes typing

)

f2\_input = widgets.IntText(

    value=1100,                     # Default value for f2 in Hz

    description='f2 (Hz):',         # Label for the widget

    continuous\_update=False         # Update only when the user finishes typing

)

# Define the output area for the plots

plot\_output2 = widgets.Output()

# Define the function to process the signal when the frequencies change

def process\_signal(change):

    """

    This function processes the signal using the specified frequencies f1 and f2.

    It designs a bandpass filter, applies it to the signal, and plots the results.

    """

    # Start the timer and show the loading animation

    loader\_html7.value = loading

    start\_time = time.time()

    # Retrieve the current values of f1 and f2 from the input widgets

    f1, f2 = f1\_input.value, f2\_input.value

    # Check if the frequencies are within acceptable limits (0 to 4000 Hz)

    if any(freq < 0 for freq in [f1, f2]) or any(freq > 4000 for freq in [f1, f2]):

        # Clear the output and display a warning message if frequencies are out of bounds

        with plot\_output2:

            plot\_output2.clear\_output(wait=True)

            warning\_html1 = widgets.HTML(

                value="<div style='color: black; background-color: #ffffcc; padding: 10px; border-radius: 5px; font-size: 16px; text-align: center;'>"

                      "<b>⚠️ Warning:</b> Frequency should not exceed 4000 and must be non-negative!</div>",

                placeholder='',

                description='',

            )

            display(warning\_html1)

            # Update the elapsed time and show completion

            elapsed\_time = time.time() - start\_time

            timer\_html7.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

            loader\_html7.value = done

    else:

        # Proceed with the processing and plotting if frequencies are valid

        with plot\_output2:

            plot\_output2.clear\_output(wait=True)  # Clear previous output

            plt.close('all')  # Close existing figures to prevent memory leaks

            # Load the signal data from the specified URL

            url = "https://raw.githubusercontent.com/ntua-el17840/Interactive-Digital-Communications/main/test-book/\_static/sima.txt"

            response = requests.get(url)

            # Convert the response text into a NumPy array of floats

            s = np.array([float(line) for line in response.text.splitlines()])

            Fs = 8192  # Sampling frequency in Hz

            # Calculate parameters for the bandpass filter design

            f2m1 = f2 - f1          # Bandwidth of the filter

            f2p1 = (f2 + f1) / 2    # Center frequency of the filter

            N = 256                 # Number of filter coefficients (taps)

            Ts = 1 / Fs             # Sampling interval

            # Time vector centered around zero for symmetric filter

            t = np.arange(-(N - 1), N, 2) \* Ts / 2

            # Design the bandpass filter using the sinc function and cosine modulation

            hbp = 2 / Fs \* np.cos(2 \* np.pi \* f2p1 \* t) \* np.sinc(f2m1 \* t)

            # Apply a Kaiser window to the filter coefficients to reduce side lobes

            hbpw = hbp \* kaiser\_atten(len(hbp), 5)

            # Compute frequency responses of the filters

            w, h = freqz(hbp, worN=8000)        # Frequency response of the basic bandpass filter

            w\_w, h\_w = freqz(hbpw, worN=8000)   # Frequency response of the windowed bandpass filter

            fs = Fs  # Sampling frequency for plotting

            # Create a figure with a 1x2 subplot layout for impulse and frequency responses

            fig, axs = plt.subplots(1, 2, figsize=(14, 4))

            # Plot the impulse responses on the first subplot

            axs[0].plot(hbp, label='Bandpass Filter', color='b')

            axs[0].plot(hbpw, label='Windowed Bandpass Filter', color='r')

            axs[0].set\_title('Impulse Responses')

            axs[0].set\_xlabel('Sample Index')

            axs[0].set\_ylabel('Amplitude')

            axs[0].legend()

            axs[0].grid(True)

            # Plot the frequency responses on the second subplot

            axs[1].plot(w / np.pi \* (fs / 2), 20 \* np.log10(abs(h)), label='Bandpass Filter', color='b')

            axs[1].plot(w\_w / np.pi \* (fs / 2), 20 \* np.log10(abs(h\_w)), label='Windowed Bandpass Filter', color='r')

            axs[1].set\_title('Frequency Responses')

            axs[1].set\_xlabel('Frequency [Hz]')

            axs[1].set\_ylabel('Magnitude [dB]')

            axs[1].legend()

            axs[1].grid(True)

            # Convolve the original signal with the filter coefficients to apply the filters

            sima\_bp = convolve(s, hbp, mode='same')    # Output of basic bandpass filter

            sima\_bpw = convolve(s, hbpw, mode='same')  # Output of windowed bandpass filter

            # Create another figure with a 1x2 subplot layout for PSD plots

            fig, axs = plt.subplots(1, 2, figsize=(14, 4))

            # Plot the Power Spectral Density (PSD) of the basic bandpass filtered signal

            f, Pxx = welch(sima\_bp, Fs, nperseg=1024)

            axs[0].semilogy(f, Pxx)

            axs[0].set\_title('PSD of Bandpass Filtered Signal')

            axs[0].set\_xlabel('Frequency [Hz]')

            axs[0].set\_ylabel('PSD [V\*\*2/Hz]')

            axs[0].grid(True)

            # Plot the PSD of the windowed bandpass filtered signal

            f, Pxx = welch(sima\_bpw, Fs, nperseg=1024)

            axs[1].semilogy(f, Pxx)

            axs[1].set\_title('PSD of Windowed Bandpass Filtered Signal')

            axs[1].set\_xlabel('Frequency [Hz]')

            axs[1].set\_ylabel('PSD [V\*\*2/Hz]')

            axs[1].grid(True)

            plt.tight\_layout()  # Adjust the layout to prevent overlap

            # Update the elapsed time and show the completion checkmark

            elapsed\_time = time.time() - start\_time

            timer\_html7.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

            loader\_html7.value = done

            plt.show()

# Attach the event handler to the frequency input widgets

# The 'process\_signal' function will be called whenever the value changes

f1\_input.observe(process\_signal, names='value')

f2\_input.observe(process\_signal, names='value')

# Arrange the input widgets in a vertical box

input\_ui = widgets.VBox([f1\_input, f2\_input])

# Group the loader and timer widgets vertically

loader\_timer\_box = widgets.VBox(

    [loader\_html7, timer\_html7],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Arrange the input widgets and loader/timer horizontally

ui = widgets.HBox(

    [input\_ui, loader\_timer\_box],

    layout=Layout(align\_items='center')

)

# Display the UI components and the plot output area

display(ui, plot\_output2)

# Call 'process\_signal' initially to display the plots with default values

process\_signal(None)

Σχεδιασμός Ζωνοπερατού Φίλτρου Parks-McClellan με Απόσβεση 60 dB και Εφαρμογή σε Σήμα Πολλαπλών Συχνοτήτων

Fs = 8192  # Sampling frequency in Hz

f1, f2 = 600, 1100  # Passband frequencies in Hz

delta\_f = 50  # Transition band width to ensure desired attenuation

# Define the frequency bands for the filter

# The structure is [0, f\_stop1, f\_pass1, f\_pass2, f\_stop2, Fs/2]

bands = [0, f1 - delta\_f, f1, f2, f2 + delta\_f, 0.5 \* Fs]

# Desired gain values in the bands: 0 for stopbands and 1 for passbands

desired = [0, 1, 0]

# Define the weights for each band to emphasize attenuation in the stopbands

weights = [100, 1, 100]

# Estimate the number of taps (filter order). May need adjustment for optimization.

numtaps = 400

# Design the FIR filter using the remez function (Parks-McClellan algorithm)

filter\_taps = remez(numtaps, bands, desired, weight=weights, fs=Fs)

# Calculate the frequency response of the filter

w, h = freqz(filter\_taps, worN=8000, fs=Fs)

# Display the frequency response

plt.figure(figsize=(14, 4))

plt.plot(w, 20 \* np.log10(abs(h)), label="Bandpass Filter")

plt.title('Frequency Response')

plt.xlabel('Frequency [Hz]')

plt.ylabel('Magnitude [dB]')

plt.grid(True)

plt.axhline(-60, color='red', linestyle='--', label="60 dB Attenuation", linewidth=1)

plt.legend()

plt.show()

Σχεδιασμός και Υλοποίηση Φίλτρου με Δύο Ζώνες Διέλευσης: (0 Hz, 500 Hz) και (1100 Hz, 3000 Hz)

Fs = 8192  # Sampling frequency in Hz

# Define our target frequency response.

# We want to create a bandpass filter that passes frequencies between 0-500 Hz and 1100-3000 Hz.

# 'target\_freqs' specifies the frequency band edges.

target\_freqs = [0, 400, 500, 1100, 1200, 2900, 3000, Fs / 2]

# 'target\_response' specifies the desired gain at each frequency in 'target\_freqs'.

# A value of 1 indicates the passband, and 0 indicates the stopband.

target\_response = [1, 1, 0, 0, 1, 1, 0, 0]

# Define the number of taps (filter order + 1) for the FIR filter.

# Increasing 'numtaps' can improve the filter's frequency response but also increases computation.

numtaps = 350  # Adjust the filter order as needed

# Design the FIR filter using the firwin2 function.

# 'firwin2' creates a linear-phase FIR filter with the specified frequency response.

filter\_taps = firwin2(numtaps, target\_freqs, target\_response, fs=Fs)

# Calculate the frequency response of the filter using 'freqz'.

# 'w' contains the frequencies at which the response is computed.

# 'h' contains the complex frequency response.

w, h = freqz(filter\_taps, worN=2000, fs=Fs)

# Display the frequency response of the filter.

plt.figure(figsize=(14, 4))

plt.plot(w, 20 \* np.log10(abs(h)), label="Bandpass Filter with firwin2")

plt.title('Frequency Response')

plt.xlabel('Frequency [Hz]')

plt.ylabel('Magnitude [dB]')

plt.grid(True)

# Add a horizontal line at -60 dB to indicate the 60 dB attenuation level.

plt.axhline(-60, color='red', linestyle='--', label="60 dB Attenuation", linewidth=1)

# Add vertical lines to mark the critical frequencies (band edges).

plt.axvline(500, color='green', linestyle='--', label="500 Hz", linewidth=1)

plt.axvline(1100, color='green', linestyle='--', label="1100 Hz", linewidth=1)

plt.axvline(3000, color='green', linestyle='--', label="3000 Hz", linewidth=1)

plt.legend()

plt.show()

### Lab Exercise 3: Προσαρμοσμένα Φίλτρα και Διαμόρφωση L-ASK

Τροποποίηση Κώδικα για Τυχαία Δειγματοληψία από Κατανεμημένες Τιμές και Επαλήθευση Ομοιόμορφης Κατανομής

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

    </div>

    """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html1 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html1 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

def generate\_histogram(k, d):

    """

    This function generates a histogram based on user-specified parameters 'k' and 'd'.

    Parameters:

    - k (int): Determines the number of quantization levels L = 2^k.

    - d (float): The distance between quantization levels.

    The function performs the following steps:

    1. Generates M random samples uniformly distributed over the quantization levels.

    2. Calculates the appropriate bins for the histogram based on 'L' and 'd'.

    3. Plots the histogram showing the frequency of each quantized value.

    4. Displays a loading animation while processing and shows the elapsed time after completion.

    """

    # Start the timer and show the loading animation

    loader\_html1.value = loading

    start\_time = time.time()

    M = 40000       # Number of random samples to generate

    L = 2 \*\* k      # Calculate L as 2 raised to the power of k, determining the number of levels

    # Generate M random samples uniformly distributed over the quantization levels

    # The quantization levels are centered around zero and spaced by 'd'

    # x will contain values like -L\*d/2 + d/2, -L\*d/2 + 3d/2, ..., L\*d/2 - d/2

    x = (2 \* np.floor(L \* np.random.rand(M)) - L + 1) \* d / 2

    # Define the bins for the histogram

    bins = np.arange(-L\*d/2, L\*d/2 + d, d)      # Bin edges from -L\*d/2 to L\*d/2 with step size 'd'

    A = np.arange(-L\*d/2 + d/2, L\*d/2, d)       # Positions for x-ticks at the center of each bin

    # Create a figure and axis for the histogram

    fig, ax = plt.subplots(1, 1, figsize=(14, 4))

    # Plot the histogram of x with specified bins and visual style

    ax.hist(x, bins=bins, edgecolor='white', color='#1F77B4')

    ax.set\_xticks(A)                    # Set the x-ticks to the centers of the bins

    ax.set\_xlabel("Integers")           # Label for the x-axis

    ax.set\_ylabel("Frequency")          # Label for the y-axis

    ax.set\_title("Histogram of array x elements")  # Title of the plot

    # Calculate and display the elapsed time

    elapsed\_time = time.time() - start\_time

    timer\_html1.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html1.value = done           # Replace the loading animation with the completion checkmark

    plt.show()                          # Display the plot

# UI Components for user input

k\_input = widgets.IntText(

    value=3,                        # Default value for k

    description='k:',               # Label for the input widget

    continuous\_update=False         # Update only when the user presses Enter or moves away from the field

)

d\_input = widgets.FloatText(

    value=1,                        # Default value for d

    description='d:',               # Label for the input widget

    continuous\_update=False         # Update only when the user presses Enter or moves away from the field

)

inputs = widgets.VBox([k\_input, d\_input])  # Arrange the input widgets vertically

# Group the loader and timer together; they will appear next to each other horizontally

loader\_timer\_box = widgets.VBox(

    [loader\_html1, timer\_html1],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')  # Add left margin and auto width

)

# Arrange the input widgets and loader/timer horizontally

ui = widgets.HBox(

    [inputs, loader\_timer\_box],

    layout=Layout(align\_items='center')  # Center the items vertically

)

# Create an interactive output that updates when 'k' or 'd' changes

out = widgets.interactive\_output(generate\_histogram, {'k': k\_input, 'd': d\_input})

# Display the UI components and the output area

display(ui, out)

Ανάλυση Ιστογραμμάτων για Διαφορετικές Τιμές EbNo και Ερμηνεία Διαφορών

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     # Start rotation at 0 degrees

        100% { transform: rotate(360deg); } # Full rotation (360 degrees)

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html2 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html2 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

def generate\_histogram(k, EbN0\_db, d):

    """

    This function generates a histogram of a noisy signal after passing through a matched filter.

    Parameters:

    - k (int): Number of bits per symbol (determines the modulation order L = 2^k)

    - EbN0\_db (float): Energy per bit to noise power spectral density ratio in decibels

    - d (float): Minimum distance between signal constellation points

    The function performs the following steps:

    1. Generates random symbols according to the specified modulation order.

    2. Adds white Gaussian noise to the upsampled signal based on the specified Eb/N0.

    3. Passes the noisy signal through a matched filter.

    4. Plots the histogram of the matched filter output.

    """

    # Start timer and show the loading animation

    loader\_html2.value = loading

    start\_time = time.time()

    M = 60000    # Number of symbols to simulate

    nsamp = 16   # Oversampling factor (number of samples per symbol)

    L = 2 \*\* k   # Modulation order (number of signal levels)

    # Calculate SNR in dB based on Eb/N0 and oversampling factor

    SNR\_db = EbN0\_db - 10 \* np.log10(nsamp / (2 \* k))

    SNR = 10 \*\* (SNR\_db \* 0.1)  # Convert SNR from dB to linear scale

    # Generate random symbols from the constellation points

    x = (2 \* np.floor(L \* np.random.rand(M)) - L + 1) \* d / 2

    # Calculate theoretical power of the signal

    Px = (d \*\* 2 / 4) \* (L \*\* 2 - 1) / 3

    # Measure the actual power of the generated symbols

    Measured\_x = np.sum(x \*\* 2) / len(x)

    # Upsample the signal by repeating each symbol nsamp times

    y = np.repeat(x, nsamp)

    # Generate white Gaussian noise with calculated variance based on SNR

    noise = np.random.normal(0, np.sqrt(Measured\_x / SNR), len(y))

    y\_noisy = y + noise  # Add noise to the upsampled signal

    # Reshape the noisy signal into a matrix where each row corresponds to one symbol

    y\_reshaped = np.reshape(y\_noisy, (M, nsamp))

    matched = np.ones((nsamp, 1))  # Matched filter coefficients (all ones)

    # Apply the matched filter to each symbol by matrix multiplication

    z = np.matmul(y\_reshaped, matched) / nsamp  # Average over nsamp samples

    # Define the positions for x-ticks based on the constellation points

    A = np.arange(-(L - 1) \* d / 2, L \* d / 2, d)  # Correct range for ticks

    # Plot the histogram of the matched filter output

    fig, ax = plt.subplots(1, 1, figsize=(14, 4))

    ax.hist(z, bins=200, edgecolor='white', color='#1F77B4')

    ax.set\_xticks(A)

    ax.set\_xlabel("Integers")

    ax.set\_ylabel("Frequency")

    ax.legend(["Eb/N0 = " + str(EbN0\_db)])

    ax.set\_title('Histogram of the Noisy Signal')

    # Calculate and display the elapsed time

    elapsed\_time = time.time() - start\_time

    timer\_html2.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html2.value = done  # Replace the loading animation with the completion checkmark

    plt.show()  # Display the plot

# UI Components for user input

k\_input = widgets.IntText(

    value=3,                        # Default value for k

    description='k:',               # Label for the input widget

    continuous\_update=False         # Update only when the user presses Enter or moves away from the field

)

d\_input = widgets.IntText(

    value=5,                        # Default value for d

    description='d:',               # Label for the input widget

    continuous\_update=False         # Update only when the user presses Enter or moves away from the field

)

EbN0\_db\_slider = widgets.FloatSlider(

    value=12,                       # Default value for Eb/N0 in dB

    min=0,

    max=20,

    step=0.1,

    description='Eb/N0 (dB):',      # Label for the slider

    continuous\_update=False         # Update only when the user releases the slider

)

# Arrange the input widgets vertically

inputs = widgets.VBox([k\_input, d\_input, EbN0\_db\_slider])

# Group the loader and timer together; they will appear next to each other horizontally

loader\_timer\_box = widgets.VBox(

    [loader\_html2, timer\_html2],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')  # Add left margin and auto width

)

# Arrange the inputs and loader/timer horizontally

ui = widgets.HBox(

    [inputs, loader\_timer\_box],

    layout=Layout(align\_items='center')  # Center the items vertically

)

# Create an interactive output that updates when any input changes

out = widgets.interactive\_output(

    generate\_histogram,

    {'k': k\_input, 'd': d\_input, 'EbN0\_db': EbN0\_db\_slider}

)

# Display the UI components and the output area

display(ui, out)

Υπολογισμός και Σύγκριση BER μέσω Εξίσωσης και BERTOOL στο MATLAB

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html3 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html3 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

def ask\_errors(k, M, nsamp, EbN0\_db):

    """

    This function simulates an ASK (Amplitude Shift Keying) communication system and calculates the number of errors.

    Parameters:

    - k (int): Number of bits per symbol (determines the modulation order L = 2^k)

    - M (int): Number of symbols to simulate

    - nsamp (int): Number of samples per symbol (oversampling factor)

    - EbN0\_db (float): Energy per bit to noise power spectral density ratio in decibels

    The function performs the following steps:

    1. Generates random symbols according to the specified modulation order.

    2. Modulates the symbols using ASK modulation.

    3. Adds white Gaussian noise to the upsampled signal based on the specified Eb/N0.

    4. Passes the noisy signal through a matched filter.

    5. Demodulates the signal and counts the number of symbol errors.

    Returns:

    - errors (int): The number of symbol errors detected.

    """

    # Calculate the modulation order

    L = 2 \*\* k

    # Adjust SNR based on the oversampling factor and bits per symbol

    SNR\_db = EbN0\_db - 10 \* np.log10(nsamp / (2 \* k))

    SNR = 10 \*\* (SNR\_db \* 0.1)  # Convert SNR from dB to linear scale

    # Generate random symbols from the constellation points

    x = 2 \* np.floor(L \* np.random.rand(M)) - L + 1  # Symbols from -L+1 to L-1, step 2

    # Calculate theoretical power of the signal

    P\_x = (L \*\* 2 - 1) / 3

    # Measure the actual power of the generated symbols

    Measured\_x = np.sum(x \*\* 2) / len(x)

    # Upsample the signal by repeating each symbol nsamp times

    y = []

    for i in range(len(x)):

        y.extend([x[i]] \* nsamp)

    y = np.array(y)

    # Generate white Gaussian noise with calculated variance based on SNR

    noise = np.random.normal(0, np.sqrt(Measured\_x / SNR), len(y))

    y\_noisy = y + noise  # Add noise to the upsampled signal

    # Reshape the noisy signal into a matrix where each row corresponds to one symbol

    y = np.reshape(y\_noisy, (M, nsamp))

    # Define the matched filter (simple averaging filter)

    matched = np.ones((nsamp, 1))

    # Apply the matched filter to each symbol by matrix multiplication

    z = np.matmul(y, matched) / nsamp  # Average over nsamp samples

    # Possible symbol levels in the modulation scheme

    l = np.arange(-L + 1, L, 2)  # Symbol levels from -L+1 to L-1, step 2

    # Flatten z to 1D array

    z = z[:, 0]

    errors = 0  # Initialize error counter

    # Demodulate and count errors

    for i in range(len(z)):

        differences = np.abs(l - z[i])  # Compute differences to all possible symbol levels

        m = np.min(differences)         # Find the minimum difference

        index = np.where(differences == m)[0][0]  # Get the index of the closest symbol

        z[i] = l[index]                 # Assign the closest symbol

        if x[i] != z[i]:                # Compare with transmitted symbol

            errors += 1                 # Increment error counter if symbols differ

    return errors  # Return the total number of errors

# Update Eb/N0 dB range to go up to 20 dB

M = 10000                 # Number of symbols to simulate

nsamp = 16                # Oversampling factor

EbN0\_db = np.arange(0, 21, 2)  # Eb/N0 values from 0 to 20 dB in steps of 2 dB

EbN0 = 10 \*\* (EbN0\_db / 10)    # Convert Eb/N0 from dB to linear scale

# Define the checkboxes for each modulation level

checkbox\_4qam = Checkbox(value=True, description='4-ASK')   # 4-level ASK modulation

checkbox\_8qam = Checkbox(value=False, description='8-ASK')  # 8-level ASK modulation

checkbox\_16qam = Checkbox(value=False, description='16-ASK')# 16-level ASK modulation

# Create an output widget to display the plot

plot\_output = Output()

def plot\_selected\_modulations(change):

    """

    This function plots the theoretical and experimental Bit Error Rate (BER) curves

    for selected ASK modulation schemes based on the checkboxes.

    Parameters:

    - change: An event handler parameter (not used in the function body)

    The function performs the following steps:

    1. Iterates over the selected modulation schemes.

    2. For each scheme, computes the experimental BER using the ask\_errors function.

    3. Computes the theoretical BER for comparison.

    4. Plots both BER curves on a semilog plot.

    """

    # Start timer

    start\_time = time.time()

    with plot\_output:

        loader\_html3.value = loading  # Display the loading animation

        plot\_output.clear\_output(wait=True)  # Clear previous output

        plt.figure(figsize=(10, 7))

        # Define colors for each modulation scheme

        colors = {'4-ASK': ('red', 'tomato'), '8-ASK': ('green', 'limegreen'), '16-ASK': ('blue', 'dodgerblue')}

        # Iterate over modulation schemes and corresponding checkboxes

        for k, checkbox in zip([2, 3, 4], [checkbox\_4qam, checkbox\_8qam, checkbox\_16qam]):

            if checkbox.value:

                L = 2 \*\* k  # Modulation order

                modulation\_name = f'{L}-ASK'

                # Compute experimental BER by simulating errors at different Eb/N0 values

                ber = [ask\_errors(k, M, nsamp, db) / (M \* k) for db in EbN0\_db]

                plt.semilogy(EbN0\_db, ber, 'o', label=f'Experimental {modulation\_name}', color=colors[modulation\_name][0])

                # Compute theoretical BER for comparison

                ber\_theoretical = (((L - 1) / L) \* erfc(np.sqrt(EbN0 \* (3 \* k) / (L \*\* 2 - 1)))) / k

                plt.semilogy(EbN0\_db, ber\_theoretical, linestyle='-', label=f'Theoretical {modulation\_name}', color=colors[modulation\_name][1])

        plt.grid(True, which='both')  # Add grid lines

        plt.xlabel("Eb/N0 (dB)")       # Label for x-axis

        plt.ylabel("BER")              # Label for y-axis

        plt.legend()                   # Display legend

        plt.title('Theoretical and Experimental BER of ASK Modulations')  # Plot title

        # Calculate and display the elapsed time

        elapsed\_time = time.time() - start\_time

        timer\_html3.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html3.value = done  # Replace the loading animation with the completion checkmark

        plt.show()

# Attach the update\_plot function to the 'value' property of each checkbox

checkbox\_4qam.observe(plot\_selected\_modulations, names='value')

checkbox\_8qam.observe(plot\_selected\_modulations, names='value')

checkbox\_16qam.observe(plot\_selected\_modulations, names='value')

# Arrange the checkboxes vertically

inputs = widgets.VBox([checkbox\_4qam, checkbox\_8qam, checkbox\_16qam])

# Group the loader and timer together; they will appear next to each other horizontally

loader\_timer\_box = widgets.VBox(

    [loader\_html3, timer\_html3],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Arrange the inputs and loader/timer horizontally

ui = widgets.HBox(

    [inputs, loader\_timer\_box],

    layout=Layout(align\_items='center')  # Center the items vertically

)

# Setup the display layout

display(ui, plot\_output)

# Display the initial plot

plot\_selected\_modulations(None)

Τροποποίηση και Επαλήθευση Κώδικα Συστήματος Διαμόρφωσης με Χρήση Φίλτρου και Προσθήκη Θορύβου

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html4 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html4 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

def ask\_errors\_new(k, M, nsamp, EbN0\_db):

    """

    This function simulates an ASK (Amplitude Shift Keying) communication system with pulse shaping

    and matched filtering, and calculates the number of symbol errors.

    Parameters:

    - k (int): Number of bits per symbol (determines the modulation order L = 2^k)

    - M (int): Number of symbols to simulate

    - nsamp (int): Number of samples per symbol (oversampling factor)

    - EbN0\_db (float): Energy per bit to noise power spectral density ratio in decibels

    The function performs the following steps:

    1. Generates random symbols according to the specified modulation order.

    2. Applies pulse shaping using an orthogonal pulse of unit energy.

    3. Upsamples the signal by inserting zeros between symbols.

    4. Convolves the upsampled signal with the pulse shaping filter.

    5. Adds white Gaussian noise to the signal based on the specified Eb/N0.

    6. Passes the noisy signal through a matched filter.

    7. Samples the matched filter output at the symbol rate.

    8. Demodulates the signal by finding the closest symbol level.

    9. Counts the number of symbol errors by comparing transmitted and received symbols.

    Returns:

    - errors (int): The number of symbol errors detected.

    """

    # Constants and signal generation

    L = 2 \*\* k  # Modulation order (number of signal levels)

    # Adjust SNR based on the oversampling factor and bits per symbol

    SNR\_db = EbN0\_db - 10 \* np.log10(nsamp / (2 \* k))

    # Generate random symbols from the constellation points

    x = 2 \* np.floor(L \* np.random.rand(M)) - L + 1  # Symbols from -L+1 to L-1, step 2

    # Filter impulse response: orthogonal pulse of unit energy

    h = np.ones(nsamp) / np.sqrt(nsamp)  # Pulse shaping filter

    # Upsample x by inserting zeros between symbols

    y\_upsampled = np.zeros(M \* nsamp)  # Initialize upsampled signal array

    y\_upsampled[::nsamp] = x  # Insert symbols every nsamp samples

    # Convolution with the filter impulse response (pulse shaping)

    y = np.convolve(y\_upsampled, h, mode='full')[:M \* nsamp]  # Truncate to original length

    # Add AWGN (Additive White Gaussian Noise)

    signal\_power = np.mean(y \*\* 2)  # Calculate signal power

    SNR\_linear = 10 \*\* (SNR\_db / 10)  # Convert SNR from dB to linear scale

    noise\_power = signal\_power / SNR\_linear  # Calculate noise power

    # Generate white Gaussian noise

    noise = np.random.normal(0, np.sqrt(noise\_power), y.shape)

    y\_noisy = y + noise  # Add noise to the signal

    # Matched filter (matched to the pulse shaping filter)

    matched = h[::-1]  # Time-reversed version of h

    yrx = np.convolve(y\_noisy, matched, mode='full')  # Convolve received signal with matched filter

    # Sampling at the end of each symbol period

    z = yrx[nsamp - 1::nsamp][:M]  # Sample every nsamp samples, starting at nsamp - 1

    # Decision making (Demodulation)

    l = np.arange(-L + 1, L, 2)  # Possible symbol levels

    z\_decoded = np.zeros(M, dtype=int)  # Initialize decoded symbols array

    for i in range(M):

        # Find the closest symbol level to the received sample

        index = np.argmin(np.abs(l - z[i]))

        z\_decoded[i] = l[index]

    # Count symbol errors by comparing transmitted and received symbols

    errors = np.sum(x != z\_decoded)

    return errors

# Simulation parameters

M = 10000  # Number of symbols to simulate

EbN0\_db = np.arange(0, 21, 2)  # Eb/N0 values from 0 to 20 dB in steps of 2 dB

# Initialize the widgets

checkbox\_4qam1 = Checkbox(value=True, description='4-ASK')   # Checkbox for 4-level ASK modulation

checkbox\_8qam1 = Checkbox(value=False, description='8-ASK')  # Checkbox for 8-level ASK modulation

checkbox\_16qam1 = Checkbox(value=False, description='16-ASK')# Checkbox for 16-level ASK modulation

nsamp\_dropdown1 = Dropdown(

    options=[4, 8, 16, 32, 64],  # Options for samples per symbol

    value=16,  # Default value

    description='Samples per Symbol:',  # Label for the dropdown

    style={'description\_width': 'initial'}

)

# Create an output widget to display the plot

plot\_output1 = Output()

def plot\_selected\_modulations1(change):

    """

    This function plots the theoretical and experimental Bit Error Rate (BER) curves

    for selected ASK modulation schemes with pulse shaping, based on the user-selected

    number of samples per symbol.

    Parameters:

    - change: An event handler parameter (not used in the function body)

    The function performs the following steps:

    1. Retrieves the selected samples per symbol (nsamp) from the dropdown.

    2. Iterates over the selected modulation schemes.

    3. For each scheme, computes the experimental BER using the ask\_errors\_new function.

    4. Computes the theoretical BER for comparison.

    5. Plots both BER curves on a semilog plot.

    """

    # Start timer and show the loading animation

    loader\_html4.value = loading

    start\_time = time.time()

    with plot\_output1:

        plot\_output1.clear\_output(wait=True)  # Clear previous output

        nsamp = nsamp\_dropdown1.value  # Get the selected value from the dropdown

        plt.figure(figsize=(10, 7))

        # Define colors for each modulation level

        colors = {

            '4-ASK': 'red',

            '8-ASK': 'green',

            '16-ASK': 'blue'

        }

        # Iterate over modulation schemes and corresponding checkboxes

        for k, checkbox, color in zip([2, 3, 4], [checkbox\_4qam1, checkbox\_8qam1, checkbox\_16qam1], colors.values()):

            if checkbox.value:

                L = 2 \*\* k  # Modulation order

                modulation\_name = f'{L}-ASK'

                ber = np.zeros(len(EbN0\_db))  # Initialize BER array

                for index, eb\_n0 in enumerate(EbN0\_db):

                    # Simulate experimental BER

                    errors = ask\_errors\_new(k, M, nsamp, eb\_n0)

                    ber[index] = errors / (M \* np.log2(L))  # Calculate BER

                # Plot experimental BER

                plt.semilogy(EbN0\_db, ber, 'o', label=f'Experimental {modulation\_name}', color=color)

                # Theoretical BER for comparison

                EbN0\_linear = 10 \*\* (EbN0\_db / 10)  # Convert Eb/N0 from dB to linear scale

                ber\_theoretical = (((L - 1) / L) \* erfc(np.sqrt(EbN0\_linear \* (3 \* np.log2(L)) / (L \*\* 2 - 1)))) / k

                plt.semilogy(EbN0\_db, ber\_theoretical, linestyle='-', label=f'Theoretical {modulation\_name}', color=color)

        plt.grid(True, which='both')  # Add grid lines

        plt.xlabel("Eb/N0 (dB)")       # Label for x-axis

        plt.ylabel("BER")              # Label for y-axis

        plt.title(f'Theoretical and Experimental BER of modified ask\_errors [nsamp={nsamp}]')  # Plot title

        plt.legend()                   # Display legend

        # Calculate and display the elapsed time

        elapsed\_time = time.time() - start\_time

        timer\_html4.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html4.value = done  # Replace the loading animation with the completion checkmark

        plt.show()

# Attach the update function to observe changes in checkboxes and dropdown

checkbox\_4qam1.observe(plot\_selected\_modulations1, names='value')

checkbox\_8qam1.observe(plot\_selected\_modulations1, names='value')

checkbox\_16qam1.observe(plot\_selected\_modulations1, names='value')

nsamp\_dropdown1.observe(plot\_selected\_modulations1, names='value')

# Arrange the input widgets vertically

inputs = widgets.VBox([nsamp\_dropdown1, checkbox\_4qam1, checkbox\_8qam1, checkbox\_16qam1])

# Group the loader and timer together; they will appear next to each other horizontally

loader\_timer\_box = widgets.VBox(

    [loader\_html4, timer\_html4],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Arrange the inputs and loader/timer horizontally

ui = widgets.HBox(

    [inputs, loader\_timer\_box],

    layout=Layout(align\_items='center')  # Center the items vertically

)

# Display the widgets and the output container

display(ui, plot\_output1)

# Call the function initially to display the plot

plot\_selected\_modulations1(None)

Σχεδίαση Τμημάτων Σημάτων Διαμόρφωσης Χωρίς Θόρυβο και Ανάλυση Αποτελεσμάτων

def ask\_errors\_modified(k, M, nsamp, add\_noise=False):

    """

    This function simulates a digital communication system with optional noise addition.

    Parameters:

    - k (int): Number of bits per symbol (determines the modulation order L = 2^k)

    - M (int): Number of symbols to simulate

    - nsamp (int): Number of samples per symbol (oversampling factor)

    - add\_noise (bool): If True, adds Additive White Gaussian Noise (AWGN) to the signal

    Returns:

    - x[:20] (ndarray): First 20 transmitted symbols

    - y[:20\*nsamp] (ndarray): First 20\*nsamp samples of the transmitted signal after pulse shaping

    - yrx[:20\*nsamp] (ndarray): First 20\*nsamp samples of the received signal after matched filtering

    """

    L = 2 \*\* k  # Modulation order (number of signal levels)

    # Generate random symbols from the constellation points

    # Symbols range from -L+1 to L-1 with a step of 2

    x = 2 \* np.floor(L \* np.random.rand(M)) - L + 1

    # Create the filter impulse response (pulse shaping filter)

    # Here, we use a rectangular pulse of duration 'nsamp' with unit energy

    h = np.ones(nsamp) / np.sqrt(nsamp)

    # Sender side: upsample and then convolve (filter) the signal

    # 'upfirdn' performs upsampling, FIR filtering, and downsampling

    y = upfirdn(h, x, up=nsamp)  # Upsample by 'nsamp' and filter with 'h'

    y = y[:M \* nsamp]  # Truncate to desired length in case convolution extends the signal

    # Optionally add noise to the transmitted signal

    y\_noisy = y

    if add\_noise:

        SNR\_db = 10  # Example SNR value in dB; adjust as necessary

        SNR\_linear = 10 \*\* (SNR\_db / 10)  # Convert SNR from dB to linear scale

        P\_x = np.mean(y \*\* 2)  # Calculate signal power

        noise\_variance = P\_x / SNR\_linear  # Calculate noise variance based on SNR

        # Generate white Gaussian noise

        noise = np.random.normal(0, np.sqrt(noise\_variance), len(y))

        y\_noisy = y + noise  # Add noise to the transmitted signal

    # Receiver side: matched filtering

    matched = h[::-1]  # Matched filter is the time-reversed version of 'h'

    yrx = convolve(y\_noisy, matched, mode='full')  # Convolve received signal with matched filter

    yrx = yrx[:M \* nsamp]  # Truncate to desired length

    # Returning slices for visualization (first 20 symbols and corresponding samples)

    return x[:20], y[:20 \* nsamp], yrx[:20 \* nsamp]

import matplotlib.pyplot as plt

k = 3         # Number of bits per symbol (e.g., k=3 for 8-ASK)

M = 100       # Total number of symbols to simulate

nsamp = 16    # Number of samples per symbol (oversampling factor)

# Get the signal parts for visualization

x\_part, y\_part, yrx\_part = ask\_errors\_modified(k, M, nsamp, add\_noise=False)

# Create a figure and a set of subplots to display the signals

fig, axs = plt.subplots(1, 3, figsize=(14, 5))

# Plot the transmitted symbols x[1:20]

markerline, stemlines, baseline = axs[0].stem(x\_part)

plt.setp(stemlines, 'linewidth', 0.5)    # Set the stem lines' width

plt.setp(markerline, 'markersize', 4)    # Set the marker size

axs[0].set\_title('x[1:20] (Transmitted Symbols)')

axs[0].set\_xlabel('Symbol Index')

axs[0].set\_ylabel('Amplitude')

# Plot the transmitted signal after pulse shaping y[1:20\*nsamp]

markerline, stemlines, baseline = axs[1].stem(y\_part)

plt.setp(stemlines, 'linewidth', 0.5)

plt.setp(markerline, 'markersize', 4)

axs[1].set\_title('y[1:20\*nsamp] (Transmitted Signal)')

axs[1].set\_xlabel('Sample Index')

axs[1].set\_ylabel('Amplitude')

# Plot the received signal after matched filtering yrx[1:20\*nsamp]

markerline, stemlines, baseline = axs[2].stem(yrx\_part)

plt.setp(stemlines, 'linewidth', 0.5)

plt.setp(markerline, 'markersize', 4)

axs[2].set\_title('yrx[1:20\*nsamp] (Received Signal)')

axs[2].set\_xlabel('Sample Index')

axs[2].set\_ylabel('Amplitude')

# Adjust layout to prevent overlap

plt.tight\_layout()

# Show the plot

plt.show()

Σύγκριση Επιδόσεων Συστήματος L-ASK με Ορθογωνικό και Συνημιτονικό Παλμό και Ανάλυση Φίλτρων για Διαφορετικές Τιμές nsamp

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html5 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html5 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

def ask\_errors\_sin\_wrong(k, M, nsamp, EbN0\_db, matched\_filter\_type='Normal'):

    """

    This function simulates an ASK communication system using a cosine-shaped pulse

    and calculates the number of symbol errors.

    Parameters:

    - k (int): Number of bits per symbol (determines the modulation order L = 2^k)

    - M (int): Number of symbols to simulate

    - nsamp (int): Number of samples per symbol (oversampling factor)

    - EbN0\_db (float): Energy per bit to noise power spectral density ratio in decibels

    - matched\_filter\_type (str): Type of matched filter to use ('Normal' or 'Reversed')

    The function performs the following steps:

    1. Generates random symbols according to the specified modulation order.

    2. Uses a cosine-shaped pulse for pulse shaping.

    3. Upsamples the signal and filters it with the pulse shape.

    4. Adds white Gaussian noise to the signal based on the specified Eb/N0.

    5. Passes the noisy signal through a matched filter.

    6. Samples the matched filter output at the symbol rate.

    7. Demodulates the signal by finding the closest symbol level.

    8. Counts the number of symbol errors.

    Returns:

    - errors (int): The number of symbol errors detected.

    """

    # Modulation order

    L = 2 \*\* k

    # Adjust SNR based on the oversampling factor and bits per symbol

    SNR\_db = EbN0\_db - 10 \* np.log10(nsamp / (2 \* k))

    SNR\_linear = 10 \*\* (SNR\_db / 10)  # Convert SNR from dB to linear scale

    # Generate random symbols from the constellation points

    x = 2 \* np.floor(L \* np.random.rand(M)) - L + 1  # Symbols from -L+1 to L-1, step 2

    # Create the pulse shaping filter using a cosine function

    # The pulse h is a cosine-shaped pulse over 'nsamp' samples

    h = np.cos(2 \* np.pi \* np.arange(1, nsamp + 1) / nsamp)

    h = h / np.sqrt(np.sum(h \*\* 2))  # Normalize the pulse to have unit energy

    # Sender side: upsample and then convolve (filter) the signal

    y = upfirdn(h, x, up=nsamp)  # Upsample by 'nsamp' and filter with 'h'

    y = y[:M \* nsamp]  # Truncate to desired length

    # Calculate signal power for noise calculation

    P\_x = np.mean(y \*\* 2)

    noise\_variance = P\_x / SNR\_linear  # Noise variance based on SNR

    # Generate white Gaussian noise

    noise = np.random.normal(0, np.sqrt(noise\_variance), len(y))

    y\_noisy = y + noise  # Add noise to the transmitted signal

    # Receiver side: matched filtering

    # Use either the normal pulse or its time-reversed version based on 'matched\_filter\_type'

    matched = h[::-1] if matched\_filter\_type == 'Reversed' else h

    yrx = np.convolve(y\_noisy, matched, mode='full')

    # Sampling at the symbol rate (every 'nsamp' samples)

    z = yrx[nsamp - 1:M \* nsamp:nsamp]

    # Decision making (Demodulation)

    levels = np.arange(-L + 1, L, 2)  # Possible symbol levels

    # For each sample, find the closest symbol level

    z\_decided = levels[np.abs(levels[:, None] - z).argmin(axis=0)]

    # Count symbol errors

    errors = np.count\_nonzero(x != z\_decided)

    return errors

# Create a RadioButtons widget for selecting the matched filter type

matched\_radio = RadioButtons(

    options=['Normal', 'Reversed'],  # Options for matched filter type

    value='Normal',  # Default value

    description='Matched Filter:',

    disabled=False

)

# Simulation parameters

M = 10000  # Number of symbols to simulate

EbN0\_db = np.arange(0, 21, 2)  # Eb/N0 values from 0 to 20 dB in steps of 2 dB

# Initialize checkboxes for selecting modulation schemes

checkbox\_4ask = Checkbox(value=True, description='4-ASK')   # Checkbox for 4-level ASK modulation

checkbox\_8ask = Checkbox(value=False, description='8-ASK')  # Checkbox for 8-level ASK modulation

checkbox\_16ask = Checkbox(value=False, description='16-ASK')# Checkbox for 16-level ASK modulation

# Dropdown widget for selecting samples per symbol

nsamp\_dropdown = Dropdown(

    options=[4, 8, 16, 32, 64],  # Options for samples per symbol

    value=16,  # Default value

    description='Samples per Symbol:',

    style={'description\_width': 'initial'}

)

# Create an output widget to display the plot

plot\_output = Output()

def plot\_selected\_modulations(change):

    """

    This function plots the theoretical and experimental Bit Error Rate (BER) curves

    for selected ASK modulation schemes using a cosine-shaped pulse, based on user-selected

    parameters such as samples per symbol and matched filter type.

    Parameters:

    - change: An event handler parameter (not used in the function body)

    The function performs the following steps:

    1. Retrieves the selected parameters from the widgets.

    2. Iterates over the selected modulation schemes.

    3. For each scheme, computes the experimental BER using 'ask\_errors\_sin\_wrong'.

    4. Computes the theoretical BER for comparison.

    5. Plots both BER curves on a semilog plot.

    """

    # Start timer and show the loading animation

    start\_time = time.time()

    loader\_html5.value = loading

    with plot\_output:

        plot\_output.clear\_output(wait=True)  # Clear previous output

        nsamp = nsamp\_dropdown.value  # Get the selected samples per symbol

        matched\_filter\_type = matched\_radio.value  # Get the selected matched filter type

        plt.figure(figsize=(10, 7))

        colors = {'4-ASK': 'red', '8-ASK': 'green', '16-ASK': 'blue'}

        # Iterate over modulation schemes and corresponding checkboxes

        for k, checkbox, color in zip(

            [2, 3, 4],  # k values for 4-ASK, 8-ASK, 16-ASK

            [checkbox\_4ask, checkbox\_8ask, checkbox\_16ask],

            colors.values()

        ):

            if checkbox.value:

                L = 2 \*\* k  # Modulation order

                modulation\_name = f'{L}-ASK'

                ber = np.zeros(len(EbN0\_db))  # Initialize BER array

                # Simulate experimental BER for each Eb/N0 value

                for index, eb\_n0 in enumerate(EbN0\_db):

                    errors = ask\_errors\_sin\_wrong(k, M, nsamp, eb\_n0, matched\_filter\_type)

                    ber[index] = errors / (M \* np.log2(L))  # Calculate BER

                # Plot experimental BER

                plt.semilogy(EbN0\_db, ber, 'o', label=f'Experimental {modulation\_name}', color=color)

                # Theoretical BER for comparison

                EbN0\_linear = 10 \*\* (EbN0\_db / 10)  # Convert Eb/N0 from dB to linear scale

                ber\_theoretical = (((L - 1) / L) \* erfc(

                    np.sqrt(EbN0\_linear \* (3 \* np.log2(L)) / (L \*\* 2 - 1))

                )) / k

                plt.semilogy(EbN0\_db, ber\_theoretical, linestyle='-', label=f'Theoretical {modulation\_name}', color=color)

        plt.grid(True, which='both')  # Add grid lines

        plt.xlabel("Eb/N0 (dB)")       # Label for x-axis

        plt.ylabel("BER")              # Label for y-axis

        plt.title(f'Theoretical and Experimental BER of ASK Modulations [nsamp={nsamp}]')

        plt.legend()                   # Display legend

        # Calculate and display the elapsed time

        elapsed\_time = time.time() - start\_time

        timer\_html5.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html5.value = done  # Replace the loading animation with the completion checkmark

        plt.show()

# Attach the update function to observe changes in checkboxes, dropdown, and radio buttons

checkbox\_4ask.observe(plot\_selected\_modulations, names='value')

checkbox\_8ask.observe(plot\_selected\_modulations, names='value')

checkbox\_16ask.observe(plot\_selected\_modulations, names='value')

nsamp\_dropdown.observe(plot\_selected\_modulations, names='value')

matched\_radio.observe(plot\_selected\_modulations, names='value')

# Arrange the widgets for user input

inputs1 = widgets.HBox([nsamp\_dropdown])  # Samples per symbol dropdown

inputs2 = widgets.HBox([matched\_radio], layout=Layout(margin="0 0 0 20px"))  # Matched filter radio buttons

inputs12 = widgets.HBox([inputs1, inputs2], layout=Layout(align\_items='center'))  # Combine inputs

inputs3 = widgets.VBox([checkbox\_4ask, checkbox\_8ask, checkbox\_16ask])  # Modulation scheme checkboxes

inputs = widgets.VBox([inputs12, inputs3])  # All input widgets

# Group the loader and timer together

loader\_timer\_box = widgets.VBox([loader\_html5, timer\_html5])

# Arrange the inputs and loader/timer horizontally

ui = widgets.HBox([inputs, loader\_timer\_box], layout=Layout(align\_items='center'))

# Display the widgets and the output container

display(ui, plot\_output)

# Call the function initially to display the plot

plot\_selected\_modulations(None)

Σχεδίαση Ορθογωνικού και Ημιτονικού Παλμού για Διάφορες Τιμές nsamp

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark symbol -->

    </div>

    """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html6 = widgets.HTML(

    value=loading  # Initially display the loading animation

)

timer\_html6 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

# Create an output widget to display the plot

output = widgets.Output()

# Define a function to plot the filters when the nsamp value changes

def plot\_filters(change):

    """

    This function plots the orthogonal and sinusoidal pulse shaping filters,

    as well as their matched filters. It updates the plots based on the selected

    number of samples per symbol (nsamp).

    Parameters:

    - change: An event handler parameter (not used in the function body)

    """

    # Start timer and show the loading animation

    start\_time = time.time()

    nsamp = nsamp\_input.value  # Get the selected nsamp value from the dropdown

    with output:

        loader\_html6.value = loading  # Display the loading animation

        output.clear\_output(wait=True)  # Clear previous output

        # Define the orthogonal pulse shaping filter

        # It is a rectangular pulse of duration 'nsamp' with unit energy

        orthogonal = np.ones(nsamp) / np.sqrt(nsamp)

        # Define the sinusoidal pulse shaping filter

        # It is a cosine function over 'nsamp' samples, normalized to unit energy

        sinusoidal = np.cos(2 \* np.pi \* np.arange(1, nsamp + 1) / nsamp)

        sinusoidal = sinusoidal / np.sqrt(np.sum(sinusoidal \*\* 2))  # Normalize to unit energy

        # Define the matched filters (time-reversed versions)

        matched\_orthogonal = orthogonal[::-1]  # Time-reversed orthogonal filter

        matched\_sinusoidal = sinusoidal[::-1]  # Time-reversed sinusoidal filter

        # Create subplots to display the filters

        fig, axs = plt.subplots(2, 2, figsize=(14, 8))

        # Plot the orthogonal filter

        markerline, stemlines, baseline = axs[0, 0].stem(orthogonal)

        plt.setp(stemlines, 'linewidth', 0.5)  # Adjust line width

        plt.setp(markerline, 'markersize', 4)  # Adjust marker size

        axs[0, 0].set\_title('Orthogonal Pulse Shaping Filter')

        axs[0, 0].set\_xlabel('Sample Index')

        axs[0, 0].set\_ylabel('Amplitude')

        # Plot the sinusoidal filter

        markerline, stemlines, baseline = axs[0, 1].stem(sinusoidal)

        plt.setp(stemlines, 'linewidth', 0.5)

        plt.setp(markerline, 'markersize', 4)

        axs[0, 1].set\_title('Sinusoidal Pulse Shaping Filter')

        axs[0, 1].set\_xlabel('Sample Index')

        axs[0, 1].set\_ylabel('Amplitude')

        # Plot the matched orthogonal filter

        markerline, stemlines, baseline = axs[1, 0].stem(matched\_orthogonal)

        plt.setp(stemlines, 'linewidth', 0.5)

        plt.setp(markerline, 'markersize', 4)

        axs[1, 0].set\_title('Matched Orthogonal Filter')

        axs[1, 0].set\_xlabel('Sample Index')

        axs[1, 0].set\_ylabel('Amplitude')

        # Plot the matched sinusoidal filter

        markerline, stemlines, baseline = axs[1, 1].stem(matched\_sinusoidal)

        plt.setp(stemlines, 'linewidth', 0.5)

        plt.setp(markerline, 'markersize', 4)

        axs[1, 1].set\_title('Matched Sinusoidal Filter')

        axs[1, 1].set\_xlabel('Sample Index')

        axs[1, 1].set\_ylabel('Amplitude')

        plt.tight\_layout()  # Adjust layout to prevent overlap

        # Calculate and display the elapsed time

        elapsed\_time = time.time() - start\_time

        timer\_html6.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html6.value = done  # Replace loading animation with completion checkmark

        plt.show()  # Display the plots

# Widget UI component: Dropdown for selecting nsamp

nsamp\_input = widgets.Dropdown(

    options=[8, 16, 32, 64],  # Possible values for nsamp

    value=8,  # Default value

    description='nsamp:',  # Label for the dropdown

)

# Attach the plot\_filters function to the nsamp\_input widget

nsamp\_input.observe(plot\_filters, names='value')

# Arrange the input widget vertically

inputs = widgets.VBox([nsamp\_input])

# Group the loader and timer together with a margin

loader\_timer\_box = widgets.VBox(

    [loader\_html6, timer\_html6],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Arrange the UI components horizontally

ui = widgets.HBox(

    [inputs, loader\_timer\_box],

    layout=Layout(align\_items='center')  # Center alignment

)

# Combine the UI and output area

out = widgets.VBox([ui, output])

# Display the user interface

display(out)

# Initial call to plot the filters with default nsamp value

plot\_filters(None)

### Lab Exercise 4: Σηματοδότηση Nyquist και Διαμόρφωση L-ASK

Παραγωγή Σήματος 8-ASK με Κωδικοποίηση Gray και Φίλτρα Nyquist: Ανάλυση Σήματος και Φάσματος

# The function grayCode accepts a number n and returns an array g containing

# the Gray coding for numbers with n bits.

# This is done by recursively calling the function gray\_code\_recurse

def grayCode(n):

    def gray\_code\_recurse (g,n):

        k=len(g)

        if n<=0:

            return

        else:

            for i in range (k-1,-1,-1):

                char='1'+g[i]

                g.append(char)

            for i in range (k-1,-1,-1):

                g[i]='0'+g[i]

            gray\_code\_recurse (g,n-1)

    g=['0','1']

    gray\_code\_recurse(g,n-1)

    return g

# The function naturalBinaryCoding accepts a number n and returns an array

# with binary numbers with n bits (or equivalently up to the number 2\*\*n, which results

# by left shifting 1 by n bits)

def naturalBinaryCoding(n):

    binary\_levels = []

    for i in range(1 << n):

        binary\_levels.append('{:0{}b}'.format(i, n))

    return binary\_levels

# The function generateRandomBits generates n\_bits random binary digits

def generateRandomBits(n\_bits):

    bitstream = []

    for i in range(n\_bits):

        random\_bit = random.randint(0, 1)

        bitstream.append(random\_bit)

    return bitstream

# The function createLevels divides the range [-A, A] into L-1 equal intervals,

# so that it always contains the endpoints of the range

def createLevels(A, L):

    y = []

    step = (A - (-A)) / (L - 1)

    for i in range(L):

        y.append(-A + i \* step)

    return y

# The function createSymbols takes an array with bits as an argument and

# groups neighboring digits into symbols with length k

def createSymbols(k, bitstream):

    n\_bits = len(bitstream)

    symbols = []

    for i in range(0, n\_bits - k + 1, k):

        symbol = ""

        for j in range(k):

            symbol += str(bitstream[i+j])

        symbols.append(symbol)

    return symbols

# The function rootRaisedCosine creates a root raised cosine pulse

# with a roll-off coefficient and order determined by the sampling rate (nsample)

# and the delay it will introduce (delay)

def rootRaisedCosine(nsamp, roll\_off, delay):

    F0 = 0.5 / nsamp

    Fd = 1

    Fs = Fd \* nsamp

    Td = 1 / Fd

    Ts = 1 / Fs

    F1 = F0 \* (1 - roll\_off)

    F2 = F0 \* (1 + roll\_off)

    filter\_order = 2 \* nsamp \* delay

    t = np.arange(0, filter\_order, Td)

    h = []

    for i in range(len(t)):

        t\_shifted = t[i] - filter\_order / 2

        if t\_shifted == 0:

            h.append(np.sqrt(2 \* F0) \*(1 + roll\_off \* ((4 / np.pi) - 1)))

        elif t\_shifted == 1 / 8 / roll\_off / F0 or t\_shifted == - 1 / 8 / roll\_off / F0 :

            h.append((roll\_off \* np.sqrt(F0)) \* ((1 + 2 / np.pi) \* np.sin(np.pi / 4 / roll\_off) + (1 - 2 / np.pi) \* np.cos(np.pi / 4 / roll\_off)))

        else:

            factor1 = np.sqrt(2 \* F0) / (1 - 64 \* roll\_off\* roll\_off \* F0 \* F0 \* t\_shifted \* t\_shifted)

            factor2 = np.sin(2 \* np.pi \* F1 \* t\_shifted) / (2 \* np.pi \* F0 \* t\_shifted)

            factor3 = (4 \* roll\_off / np.pi) \* np.cos(2 \* np.pi \* F2 \* t\_shifted)

            h.append(factor1 \* (factor2 + factor3))

    return t,h

# The function upSample increases the number of samples of a signal signal by adding nsamp-1

# zeros after each sample of the signal

def upSample(signal, nsamp):

    upSampled = []

    for i in range(len(signal) \* nsamp):

        if i % nsamp == 0:

            upSampled.append(signal[i // nsamp])

        else:

            upSampled.append(0)

    return upSampled

# The function downSample reduces the sampling frequency of a signal signal by a factor of nsamp

# by keeping only the samples that are multiples of nsamp (0, nsamp, 2\*nsamp, ...)

def downSample(signal, nsamp):

    downSampled = []

    for i in range(0, len(signal), nsamp):

        downSampled.append(signal[i])

    return downSampled

# Adds white Gaussian noise with mean value μ (mu) and variance σ^2 (sigma)

def generateAWGN(signal, mu, sigma):

    noise = sigma \* np.random.randn(len(signal)) + mu

    return noise

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html1 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html1 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

# Declare global variables for filter and parameters

global filt, g\_delay, g\_nsamp

def interactive\_signal\_processing1(n\_bits, L, roll\_off, nsamp, delay):

    """

    This function performs interactive signal processing for an L-ASK modulation scheme.

    It generates random bits, maps them to symbols using Gray encoding, applies pulse shaping

    with a root raised cosine filter, and visualizes the signals at different stages.

    Parameters:

    - n\_bits (int): Length of the binary bitstream.

    - L (int): Number of amplitude levels in ASK modulation (e.g., 2, 4, 8, 16).

    - roll\_off (float): Roll-off factor for the root raised cosine filter.

    - nsamp (int): Number of samples per symbol (oversampling factor).

    - delay (int): Group delay of the filter.

    The function updates global variables for the filter, delay, and nsamp.

    It also displays four plots:

    1. Root Raised Cosine Filter response.

    2. Filtered signal with original symbols overlaid.

    3. Transmitted and received signals for comparison.

    4. Power Spectral Density of the received signal.

    """

    # Start timer and display loading animation

    loader\_html1.value = loading

    start\_time = time.time()

    # Update global variables for use outside the function

    global filt, g\_delay, g\_nsamp

    g\_delay = delay

    g\_nsamp = nsamp

    # Generate a random bitstream of length n\_bits

    bitstream = generateRandomBits(n\_bits)

    # Calculate the amplitude (A) and bits per symbol (k)

    A = L - 1

    k = int(log2(L))

    # Create the amplitude levels for the modulation

    y\_levels = createLevels(A, L)

    # Map bits to symbols based on bits per symbol (k)

    symbols = createSymbols(k, bitstream)

    # Generate Gray encoding for symbol mapping

    gray\_encoding = grayCode(k)

    # Map the symbols to amplitude levels using Gray encoding

    x\_gray = [y\_levels[gray\_encoding.index(symbol)] for symbol in symbols]

    # Generate the root raised cosine filter

    t\_filter, filt = rootRaisedCosine(nsamp, roll\_off, delay)

    # Upsample the signal by inserting zeros between samples

    y = upSample(x\_gray, nsamp)

    # Convolve the upsampled signal with the filter (transmit filtering)

    y\_transmitted = scipy.signal.convolve(y, filt)

    # Convolve the transmitted signal with the filter again (receive filtering)

    y\_received = scipy.signal.convolve(y\_transmitted, filt)

    # Downsample the received signal to symbol rate

    y\_final = downSample(y\_received, nsamp)

    y\_final = y\_final[2 \* delay: len(y\_final) - 2 \* delay]  # Adjust for filter delay

    # Create a figure object with specified size

    fig = plt.figure(figsize=(20, 12))

    # First subplot - Root Raised Cosine Filter response

    ax1 = fig.add\_subplot(2, 2, 1)

    ax1.plot(t\_filter, filt)

    ax1.set\_title('Root Raised Cosine Filter')

    ax1.set\_xlabel('Time')

    ax1.set\_ylabel('Amplitude')

    ax1.grid(True)

    # Second subplot - Filtered signal with original symbols overlaid

    ax2 = fig.add\_subplot(2, 2, 2)

    t\_continuous = np.arange(0, len(y\_transmitted[:10\*nsamp]))  # Time vector for continuous signal

    ax2.plot(t\_continuous, y\_transmitted[:10\*nsamp], label='Filtered Signal')

    # Generate stem positions for original symbols

    t\_symbols = np.arange(0, 10\*nsamp, nsamp)  # Stem positions every nsamp samples

    y\_stems = y\_transmitted[t\_symbols]  # Corresponding y-values at stem positions

    # Overlay stem plot for original symbols

    ax2.stem(t\_symbols, y\_stems, linefmt='C1-', markerfmt='C1o', basefmt=" ", label='Original Symbols')

    ax2.set\_title('Signal Visualization')

    ax2.set\_xlabel('Time')

    ax2.set\_ylabel('Amplitude')

    ax2.legend()

    ax2.grid(True)

    # Third subplot - Transmitted and received signals comparison

    ax3 = fig.add\_subplot(2, 2, 3)

    t = np.arange(0, len(y[:10]))

    ax3.plot(t, y\_final[:10], label='Received')

    ax3.stem(t, x\_gray[:10], label='Transmitted', linefmt='C1-', markerfmt='C1o', basefmt=" ")

    ax3.set\_title('Signal Visualization')

    ax3.set\_xlabel('Time')

    ax3.set\_ylabel('Amplitude')

    ax3.legend()

    ax3.grid(True)

    # Fourth subplot - Power Spectral Density of the Received Signal

    ax4 = fig.add\_subplot(2, 2, 4)

    f, Pxx\_den = scipy.signal.welch(y\_received, window='hamming', nperseg=8192)

    Pxx\_den = 10 \* np.log10(Pxx\_den)  # Convert to dB scale

    ax4.plot(f, Pxx\_den)

    ax4.set\_title('Power Spectral Density of the Received Signal')

    ax4.set\_xlabel('Normalized Frequency')

    ax4.set\_ylabel('Power/Frequency [dB]')

    ax4.grid(True)

    plt.tight\_layout()  # Adjust the layout to prevent overlap

    plt.show()

    # Display elapsed time and update loading animation

    elapsed\_time = time.time() - start\_time

    timer\_html1.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html1.value = done

# Widget setup for interactive parameters

n\_bits\_slider1 = IntSlider(

    min=10000, max=100000, step=10000, value=10000,

    description='Bitstream Length', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

roll\_off\_slider1 = FloatSlider(

    min=0.1, max=1.0, step=0.1, value=0.4,

    description='Roll-off Factor', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

L\_dropdown1 = Dropdown(

    options=[2\*\*i for i in range(1, 6)], value=2,

    description='ASK Levels', style={'description\_width': 'initial'},

    continuous\_update=False

)

nsamp\_slider1 = IntSlider(

    min=10, max=40, step=1, value=20,

    description='nsamp', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

delay\_slider1 = IntSlider(

    min=1, max=10, step=1, value=5,

    description='Group Delay', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

# Create the interactive output for the function

out = widgets.interactive\_output(

    interactive\_signal\_processing1,

    {'n\_bits': n\_bits\_slider1,

     'L': L\_dropdown1,

     'roll\_off': roll\_off\_slider1,

     'nsamp': nsamp\_slider1,

     'delay': delay\_slider1}

)

# Function to update filter order when sliders change

def update\_filter\_order1(\*args):

    filter\_order1 = 2 \* nsamp\_slider1.value \* delay\_slider1.value

    filter\_order\_display1.value = filter\_order1

# Observers to trigger filter order update on slider value changes

nsamp\_slider1.observe(update\_filter\_order1, 'value')

delay\_slider1.observe(update\_filter\_order1, 'value')

# Display the filter order with an IntText widget

filter\_order\_display1 = widgets.IntText(

    value=2 \* nsamp\_slider1.value \* delay\_slider1.value,

    description='Filter Order:',

    disabled=True

)

# Arrange the input widgets in a VBox layout

inputs = widgets.VBox([

    n\_bits\_slider1, roll\_off\_slider1, nsamp\_slider1,

    delay\_slider1, filter\_order\_display1, L\_dropdown1

])

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox(

    [loader\_html1, timer\_html1],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Arrange the UI with HBox and layout options

ui = widgets.HBox(

    [inputs, loader\_timer\_box],

    layout=widgets.Layout(align\_items='center', flex\_flow='row nowrap')

)

inputs.layout.flex = '1 1 auto'  # Flex-grow, flex-shrink, flex-basis for input widgets

loader\_timer\_box.layout.flex = '0 1 auto'  # No flex-grow for loader/timer box

# Display the UI components and the interactive output

clear\_output(wait=True)  # Clear the previous output to refresh

display(ui, out)

Μελέτη Επίδοσης BER για 8-ASK: Επίδραση Τάξης Φίλτρου Nyquist και Roll-Off

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html2 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html2 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

def interactive\_signal\_processing2(n\_bits=80000, roll\_off=0.7, nsamp=16, delay=4, L=16):

    """

    This function simulates an L-ASK communication system with pulse shaping and additive white Gaussian noise (AWGN).

    It calculates and plots both theoretical and simulated Bit Error Rate (BER) curves.

    Parameters:

    - n\_bits (int): Length of the binary bitstream.

    - roll\_off (float): Roll-off factor for the root raised cosine filter.

    - nsamp (int): Number of samples per symbol (oversampling factor).

    - delay (int): Group delay of the filter.

    - L (int): Number of amplitude levels in ASK modulation (e.g., 2, 4, 8, 16).

    The function performs the following steps:

    1. Generates random bits and maps them to symbols using Gray encoding.

    2. Applies pulse shaping with a root raised cosine filter.

    3. Simulates transmission over an AWGN channel for various Eb/N0 values.

    4. Calculates the theoretical and simulated BER.

    5. Plots the BER curves.

    """

    # Start timer and display loading animation

    loader\_html2.value = loading

    start\_time = time.time()

    # Generate a random bitstream of length n\_bits

    bitstream = generateRandomBits(n\_bits)

    # Calculate the amplitude (A) and bits per symbol (k)

    A = L - 1

    k = int(log2(L))

    # Create the amplitude levels for the modulation

    y\_levels = createLevels(A, L)

    # Map bits to symbols based on bits per symbol (k)

    symbols = createSymbols(k, bitstream)

    # Generate Gray encoding for symbol mapping

    gray\_encoding = grayCode(k)

    # Map the symbols to amplitude levels using Gray encoding

    x\_gray = [y\_levels[gray\_encoding.index(symbol)] for symbol in symbols]

    # Generate the root raised cosine filter

    t\_filter, filt = rootRaisedCosine(nsamp, roll\_off, delay)

    # Upsample the signal by inserting zeros between samples

    y = upSample(x\_gray, nsamp)

    # Convolve the upsampled signal with the filter (transmit filtering)

    y\_transmitted = scipy.signal.convolve(y, filt)

    # Initialize lists to store BER values

    berTheoretical = []

    berSimulation = []

    EbN0\_max = 20  # Maximum Eb/N0 value in dB

    # Loop over a range of Eb/N0 values to simulate different noise levels

    for EbN0\_db in range(1, EbN0\_max):

        # Convert Eb/N0 from dB to linear scale

        EbN0 = 10 \*\* (EbN0\_db / 10)

        # Adjust SNR based on oversampling factor and bits per symbol

        SNR\_db = EbN0\_db - 10 \* np.log10(nsamp / 2 / k)

        SNR = 10 \*\* (SNR\_db / 10)

        # Calculate the power of the transmitted signal and noise power

        P = np.mean(y\_transmitted \*\* 2)

        Pn = P / SNR

        # Generate AWGN noise

        noise = np.random.normal(0, sqrt(Pn), len(y\_transmitted))

        y\_noisy = y\_transmitted + noise  # Add noise to the transmitted signal

        # Convolve the noisy signal with the filter (receive filtering)

        z\_received = scipy.signal.convolve(y\_noisy, filt)

        # Downsample the received signal to symbol rate

        z\_final = downSample(z\_received, nsamp)

        z\_final = z\_final[2 \* delay : len(z\_final) - 2 \* delay]  # Adjust for filter delay

        # Initialize error counter

        errors = 0

        # Symbol decision based on minimum distance

        for i in range(len(z\_final)):

            differences = np.abs(y\_levels - z\_final[i])

            index = np.argmin(differences)

            if y\_levels[index] != x\_gray[i]:

                errors += 1

        # Calculate simulated BER

        berSimulation.append(errors / n\_bits)

        # Calculate theoretical BER for L-ASK modulation

        Pe = (L - 1) / L \* erfc(sqrt(3 \* k / (L\*\*2 - 1) \* EbN0))

        berTheoretical.append(Pe / k)

    # Plotting the BER curves

    plt.figure(figsize=(10, 6))

    plt.semilogy(berTheoretical, label='Theoretical')

    plt.semilogy(berSimulation, 'o', label='Simulation')

    plt.xlabel('Eb/N0 (dB)')

    plt.ylabel('Bit Error Rate')

    plt.title('BER Curve for L-ASK')

    plt.legend()

    plt.grid(True)

    # Display elapsed time and update loading animation

    elapsed\_time = time.time() - start\_time

    timer\_html2.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html2.value = done

    plt.show()

# Creating sliders and dropdowns for interactive parameters

n\_bits\_slider2 = IntSlider(

    min=10000, max=100000, step=10000, value=80000,

    description='Binary Sequence Length', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

roll\_off\_slider2 = FloatSlider(

    min=0.1, max=1.0, step=0.1, value=0.7,

    description='Roll-off Factor', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

nsamp\_slider2 = IntSlider(

    min=8, max=32, step=1, value=16,

    description='nsamp', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

delay\_slider2 = IntSlider(

    min=1, max=8, step=1, value=4,

    description='Group Delay', style={'description\_width': 'initial'},

    layout=Layout(width='100%'), continuous\_update=False

)

L\_dropdown2 = Dropdown(

    options=[2\*\*i for i in range(1, 6)], value=16,

    description='ASK Levels', style={'description\_width': 'initial'},

    continuous\_update=False

)

# Create the interactive output for the function

out = widgets.interactive\_output(

    interactive\_signal\_processing2,

    {'n\_bits': n\_bits\_slider2,

     'roll\_off': roll\_off\_slider2,

     'nsamp': nsamp\_slider2,

     'delay': delay\_slider2,

     'L': L\_dropdown2}

)

# Display filter order with an IntText widget

filter\_order\_display2 = widgets.IntText(

    value=2 \* nsamp\_slider2.value \* delay\_slider2.value,

    description='Filter Order:',

    disabled=True

)

# Function to update filter order dynamically when sliders change

def update\_filter\_order2(\*args):

    filter\_order2 = 2 \* nsamp\_slider2.value \* delay\_slider2.value

    filter\_order\_display2.value = filter\_order2

# Attach observers to sliders to update filter order when values change

nsamp\_slider2.observe(update\_filter\_order2, 'value')

delay\_slider2.observe(update\_filter\_order2, 'value')

# Group the input widgets into a VBox

input\_widgets = widgets.VBox([

    n\_bits\_slider2, roll\_off\_slider2, nsamp\_slider2,

    delay\_slider2, filter\_order\_display2, L\_dropdown2

], layout=widgets.Layout(flex='1 1 auto', width='auto'))

# Group the loader and timer together

loader\_timer\_box = widgets.VBox(

    [loader\_html2, timer\_html2],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Combine input widgets and loader/timer into a horizontal layout (HBox)

ui = widgets.HBox(

    [input\_widgets, loader\_timer\_box],

    layout=widgets.Layout(align\_items='center')

)

# Display the UI components and the interactive output

clear\_output(wait=True)  # Clear the previous output

display(ui, out)

Σύγκριση Καμπυλών BER-Eb/No για 16-ASK και 8-ASK με Μη Gray Κωδικοποίηση

# Loading animation HTML code for visual feedback during processing

# This HTML code creates a spinning loader animation to indicate that processing is ongoing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Full rotation (360 degrees) \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

# This code shows a green checkmark indicating that processing has finished

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html3 = widgets.HTML(

  value=loading  # Initially display the loading animation

)

timer\_html3 = widgets.HTML(

    value="Elapsed time: - seconds"  # Placeholder for the elapsed time display

)

def interactive\_signal\_processing\_with\_natural(n\_bits=80000, roll\_off=0.7, nsamp=16, delay=4, L=16):

    """

    This function simulates an L-ASK communication system using both Gray and Natural encoding schemes.

    It calculates and compares the Bit Error Rate (BER) for both encoding methods over an AWGN channel.

    Parameters:

    - n\_bits (int): Length of the binary bitstream.

    - roll\_off (float): Roll-off factor for the root raised cosine filter.

    - nsamp (int): Number of samples per symbol (oversampling factor).

    - delay (int): Group delay of the filter.

    - L (int): Number of amplitude levels in ASK modulation (e.g., 2, 4, 8, 16).

    The function performs the following steps:

    1. Generates random bits and maps them to symbols using both Gray and Natural encoding.

    2. Applies pulse shaping with a root raised cosine filter.

    3. Simulates transmission over an AWGN channel for various Eb/N0 values.

    4. Calculates the BER for both encoding schemes.

    5. Plots the BER curves for comparison.

    """

    # Start timer

    loader\_html3.value = loading

    start\_time = time.time()

    # Common setup

    bitstream = generateRandomBits(n\_bits)

    A = L - 1

    k = int(log2(L))

    y\_levels = createLevels(A, L)

    symbols = createSymbols(k, bitstream)

    # Gray encoding setup

    gray\_encoding = grayCode(k)

    x\_gray = [y\_levels[gray\_encoding.index(symbol)] for symbol in symbols]

    # Natural encoding setup

    natural\_encoding = naturalBinaryCoding(k)

    x\_natural = [y\_levels[natural\_encoding.index(symbol)] for symbol in symbols]

    # Root raised cosine filter (common for both Gray and Natural)

    t\_filter, filt = rootRaisedCosine(nsamp, roll\_off, delay)

    # Transmission simulation for Gray encoding

    y\_gray = upSample(x\_gray, nsamp)

    y\_transmitted\_gray = scipy.signal.convolve(y\_gray, filt)

    # Transmission simulation for Natural encoding

    y\_natural = upSample(x\_natural, nsamp)

    y\_transmitted\_natural = scipy.signal.convolve(y\_natural, filt)

    # Calculate BER for Gray encoding

    berTheoretical = []

    berSimulation\_gray = []

    berSimulation\_natural = [] # For natural encoding

    EbN0\_max = 20

    for EbN0\_db in range(1, EbN0\_max):

        # Convert Eb/N0 from dB to linear scale

        EbN0 = 10 \*\* (EbN0\_db \* 0.1)

        SNR\_db = EbN0\_db - 10\*np.log10(nsamp/2/k)

        SNR = 10 \*\* (SNR\_db \* 0.1)

        # Calculate the power of the transmitted signal and the noise power to achieve the desired SNR

        P = sum(y\_transmitted\_gray \* y\_transmitted\_gray) / len(y\_transmitted\_gray)

        P\_db = 10 \* np.log10(P)

        Pn\_db = P\_db - SNR\_db

        Pn = 10 \*\* (Pn\_db \* 0.1)

        # Add noise (common for both Gray and Natural)

        mu = 0

        sigma = np.sqrt(Pn)

        noise = generateAWGN(y\_transmitted\_gray, mu, sigma)

        y\_noisy\_gray = y\_transmitted\_gray + noise

        y\_noisy\_natural = y\_transmitted\_natural + noise

        # Receiver simulation for Gray encoding

        z\_received\_gray = scipy.signal.convolve(y\_noisy\_gray, filt)

        z\_final\_gray = downSample(z\_received\_gray, nsamp)

        z\_final\_gray = z\_final\_gray[2 \* delay : len(z\_final\_gray) - 2 \* delay]

        # Receiver simulation for Natural encoding (Code 1)

        z\_received\_natural = scipy.signal.convolve(y\_noisy\_natural, filt)

        z\_final\_natural = downSample(z\_received\_natural, nsamp)

        z\_final\_natural = z\_final\_natural[2 \* delay : len(z\_final\_natural) - 2 \* delay]

        # Error calculation for Gray encoding

        # Decision for the level corresponding to the symbol received for Gray encoding

        for i in range(len(z\_final\_gray)):

            # Array with the differences of the signal from the levels

            differences = np.abs(y\_levels - z\_final\_gray[i])

            m = min(differences) # Find the minimum distance

            [index], = np.where(differences == m)

            z\_final\_gray[i] = y\_levels[index]

        # We assume that each error in Gray encoding results in one incorrect bit

        error = 0

        for i in range(len(z\_final\_gray)):

            if x\_gray[i] != z\_final\_gray[i]:

                error += 1

        # BER calculation = number of errors / number of bits

        berSimulation\_gray.append(error/n\_bits)

        # Error calculation for Natural encoding

        # Decision for the level corresponding to the symbol received

        for i in range(len(z\_final\_natural)):

            differences = np.abs(y\_levels - z\_final\_natural[i])

            m = min(differences)

            [index], = np.where(differences == m)

            z\_final\_natural[i] = y\_levels[index]

        # The final\_symbols contains the bits that were decided to be received

        final\_symbols = []

        for i in range(len(z\_final\_natural)):

            # Mapping each decided level to its binary encoding

            index = y\_levels.index(z\_final\_natural[i])

            final\_symbols.append(natural\_encoding[index])

        # For each incorrect symbol, we need to check how many bits were wrong,

        # as the encoding of neighboring levels no longer differs by only one bit

        error = 0

        for i in range(len(z\_final\_natural)):

            # If a symbol is wrong, check how many digits were received incorrectly

            if x\_natural[i] != z\_final\_natural[i]:

                for j in range(len(symbols[i])):

                    if symbols[i][j] != final\_symbols[i][j]:

                        error += 1

        berSimulation\_natural.append(error/n\_bits)

        # Theoretical BER calculation (common for both Gray and Natural)

        Pe = (L - 1) / L \* erfc(sqrt(3 \* k / (L\*\*2 - 1) \* EbN0))

        berTheoretical.append(Pe / k)

    # Plotting the BER for both Gray and Natural encoding

    plt.figure(figsize=(10, 6))

    plt.semilogy(range(1, EbN0\_max), berTheoretical, label='Theoretical')

    plt.semilogy(range(1, EbN0\_max), berSimulation\_gray, 'o', label='Gray')

    plt.semilogy(range(1, EbN0\_max), berSimulation\_natural, '\*', label='Natural')

    plt.xlabel('Eb/N0 (dB)')

    plt.ylabel('Bit Error Rate')

    plt.title('BER Curve for L-ASK')

    plt.legend()

    plt.grid(True)

    # Show elapsed time

    elapsed\_time = time.time() - start\_time

    timer\_html3.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html3.value = done

    plt.show()

# Creating sliders and dropdown

n\_bits\_slider1 = IntSlider(min=10000, max=100000, step=10000, value=80000, description='Binary Sequence Length', style={'description\_width': 'initial'}, layout=Layout(width='100%'), continuous\_update=False)

roll\_off\_slider1 = FloatSlider(min=0.1, max=1.0, step=0.1, value=0.7, description='Roll-off Factor', style={'description\_width': 'initial'}, layout=Layout(width='100%'), continuous\_update=False)

nsamp\_slider1 = IntSlider(min=8, max=32, step=1, value=16, description='nsamp', style={'description\_width': 'initial'}, layout=Layout(width='100%'), continuous\_update=False)

delay\_slider1 = IntSlider(min=1, max=8, step=1, value=4, description='Group Delay', style={'description\_width': 'initial'}, layout=Layout(width='100%'), continuous\_update=False)

L\_dropdown1 = Dropdown(options=[2\*\*i for i in range(1, 6)], value=16, description='ASK Levels', style={'description\_width': 'initial'}, continuous\_update=False)

# Create a VBox for the input widgets (similar to the first code snippet)

input\_widgets = widgets.VBox([n\_bits\_slider1, roll\_off\_slider1, nsamp\_slider1, delay\_slider1, L\_dropdown1], layout=widgets.Layout(flex='1 1 auto', width='auto'))

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox([loader\_html3, timer\_html3], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Combine input widgets and loader/timer box into an HBox with aligned items

ui = widgets.HBox([input\_widgets, loader\_timer\_box], layout=widgets.Layout(align\_items='center'))

# Create the interactive output for the interactive\_signal\_processing\_with\_natural function

out = widgets.interactive\_output(interactive\_signal\_processing\_with\_natural,

                                 {'n\_bits': n\_bits\_slider1,

                                  'roll\_off': roll\_off\_slider1,

                                  'nsamp': nsamp\_slider1,

                                  'delay': delay\_slider1,

                                  'L': L\_dropdown1})

# Display the UI and output

clear\_output(wait=True)  # Clear the previous output

display(ui, out)

### Lab Exercise 5: Διαμόρφωση QAM και PSK

Σχεδίαση Σηματικού Αστερισμού 64-QAM με Κωδικοποίηση Gray

# Loading animation HTML code for visual feedback during processing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'>

        </div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation \*/

        100% { transform: rotate(360deg); } /\* End rotation \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Checkmark symbol -->

    </div>

    """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html1 = widgets.HTML(value=loading)  # Initially show loading animation

timer\_html1 = widgets.HTML(value="Elapsed time: - seconds")  # Placeholder for elapsed time

def generate\_qam\_constellation(L):

    """

    Generate the QAM constellation points for a square L x L QAM.

    Parameters:

    - L: Number of points along one axis (e.g., L=4 for 16-QAM)

    Returns:

    - mapping: 1D numpy array of complex numbers representing constellation points

    """

    M = L \* L                  # Total number of constellation points

    l = int(np.log2(L))        # Number of bits per axis

    # Generate equally spaced points along the real and imaginary axes

    x = np.arange(-(L - 1), L, 2)

    y = np.arange(-(L - 1), L, 2)

    # Create a grid of points (constellation)

    xv, yv = np.meshgrid(x, y)

    mapping = xv + 1j \* yv     # Combine real and imaginary parts

    mapping = mapping.flatten()  # Flatten to 1D array

    return mapping

def plot\_qam\_constellation(L):

    """

    Plot the QAM constellation for a given L.

    Parameters:

    - L: Number of points along one axis (e.g., L=4 for 16-QAM)

    """

    # Start timer and display loading animation

    loader\_html1.value = loading

    start\_time = time.time()

    mapping = generate\_qam\_constellation(L)

    M = L \* L

    l = int(np.log2(L))

    # Plot the constellation points

    plt.figure(figsize=(10, 7))

    plt.scatter(mapping.real, mapping.imag)

    if L < 16:  # Include labels for smaller constellations

        # Generate binary labels for each point

        labels = [bin(i)[2:].zfill(2 \* l) for i in range(M)]

        dx, dy = -0.5, 0.3  # Offset for labels

        for i in range(len(labels)):

            plt.text(mapping[i].real + dx, mapping[i].imag + dy, labels[i],

                     bbox=dict(facecolor='red', alpha=0.5))

    plt.grid(True)

    plt.xlim(-L, L)

    plt.ylim(-L, L)

    plt.title(f"{L}x{L} QAM Constellation")

    plt.xlabel("In-phase")

    plt.ylabel("Quadrature")

    plt.show()

    # Update elapsed time and loading animation

    elapsed\_time = time.time() - start\_time

    timer\_html1.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html1.value = done

# Create a dropdown widget to select the value of L

L\_dropdown = Dropdown(options=[2, 4, 8, 16, 32, 64], value=8, description='L-QAM:', continuous\_update=False)

# Group the loader and timer widgets vertically

loader\_timer\_box = widgets.VBox([loader\_html1, timer\_html1], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Create an interactive plot that updates when L changes

interactive\_plot = interact(plot\_qam\_constellation, L=L\_dropdown)

# Arrange input widgets and output

input\_widgets = VBox([L\_dropdown], layout=Layout(width='auto'))

plot\_output = interactive\_plot.widget.children[-1]  # The output plot widget

# Combine input widgets and loader/timer into a horizontal box

inputs\_and\_loader = HBox([input\_widgets, loader\_timer\_box], layout=Layout(align\_items='center'))

# Create a vertical box to hold the entire UI

ui = VBox([inputs\_and\_loader, plot\_output])

# Clear previous output and display the UI

clear\_output(wait=True)

display(ui)

Σχεδίαση Σηματικού Αστερισμού PSK

# Loading animation HTML code for visual feedback during processing

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'>

        </div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation \*/

        100% { transform: rotate(360deg); } /\* End rotation \*/

    }

    </style>

    """

# HTML code to display a checkmark when loading is complete

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Checkmark symbol -->

    </div>

    """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html2 = widgets.HTML(value=loading)

timer\_html2 = widgets.HTML(value="Elapsed time: - seconds")

def generate\_psk\_constellation(M):

    """

    Generate the PSK constellation points for M-PSK modulation.

    Parameters:

    - M: Modulation order (e.g., M=8 for 8-PSK)

    Returns:

    - mapping: Numpy array of complex numbers representing constellation points

    """

    k = int(np.log2(M))  # Number of bits per symbol

    ph1 = np.pi / 4      # Initial phase

    # Initialize theta with initial phase values

    theta = np.array([ph1, -ph1, np.pi - ph1, -np.pi + ph1])

    mapping = np.exp(1j \* theta)  # Initial constellation points

    # Generate PSK constellation for higher-order M

    if k > 2:

        for j in range(3, k + 1):

            theta = theta / 2

            mapping = np.exp(1j \* theta)

            mapping = np.concatenate((mapping, -np.conjugate(mapping)))  # Mirror across origin

            theta = np.angle(mapping)  # Update theta

    return mapping

def plot\_psk\_constellation(M):

    """

    Plot the PSK constellation for a given M.

    Parameters:

    - M: Modulation order (e.g., M=8 for 8-PSK)

    """

    # Start timer and display loading animation

    loader\_html2.value = loading

    start\_time = time.time()

    constellation = generate\_psk\_constellation(M)

    k = int(np.log2(M))

    # Plot the constellation points

    plt.figure(figsize=(10, 7))

    plt.scatter(np.real(constellation), np.imag(constellation))

    plt.grid(True)

    plt.title(f'{M}-PSK Constellation')

    plt.xlabel('In-Phase')

    plt.ylabel('Quadrature')

    plt.axhline(0, color='gray', linewidth=0.5)  # Horizontal axis

    plt.axvline(0, color='gray', linewidth=0.5)  # Vertical axis

    # Label each point with its binary representation

    for m in range(len(constellation)):

        plt.text(np.real(constellation[m]) + 0.05, np.imag(constellation[m]),

                 format(m, '0{}b'.format(k)),

                 bbox=dict(facecolor='red', alpha=0.5))

    plt.show()

    # Update elapsed time and loading animation

    elapsed\_time = time.time() - start\_time

    timer\_html2.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html2.value = done

# Create a dropdown widget to select the value of M

M\_dropdown = widgets.Dropdown(options=[4, 8, 16, 32, 64], value=16, description='M-PSK:')

# Group the loader and timer widgets vertically

loader\_timer\_box = widgets.VBox([loader\_html2, timer\_html2], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Create an interactive plot that updates when M changes

interactive\_plot = interact(plot\_psk\_constellation, M=M\_dropdown)

# Arrange input widgets and output

input\_widgets = VBox([M\_dropdown], layout=Layout(width='auto'))

plot\_output = interactive\_plot.widget.children[-1]

# Combine input widgets and loader/timer into a horizontal box

inputs\_and\_loader = HBox([input\_widgets, loader\_timer\_box], layout=Layout(align\_items='center'))

# Create a vertical box to hold the entire UI

ui = VBox([inputs\_and\_loader, plot\_output])

# Clear previous output and display the UI

clear\_output(wait=True)

display(ui)

Επιλογή και Εξομοίωση Συστήματος M-QAM για Ρυθμό 10 Mbps σε Ζωνοπερατό Δίαυλο 6.75-9.25 MHz

# HTML loading animation and completion checkmark

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'>

        </div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation \*/

        100% { transform: rotate(360deg); } /\* End rotation \*/

    }

    </style>

    """

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Checkmark symbol -->

    </div>

    """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html3 = widgets.HTML(value=loading)

timer\_html3 = widgets.HTML(value="Elapsed time: - seconds")

# Function to calculate roll-off factor and recommend the best M

def calculate\_rolloff(f1, f2, R):

    """

    Calculate the roll-off factor and recommend the best modulation order M.

    Parameters:

    - f1: Lower frequency in MHz

    - f2: Upper frequency in MHz

    - R: Data rate in Mbps

    Returns:

    - best\_a: Calculated roll-off factor

    - best\_M: Recommended modulation order M

    """

    W = (f2 - f1) \* 1e6  # Bandwidth in Hz

    R = R \* 1e6          # Data rate in bps

    possible\_Ms = [4, 16, 64, 256, 1024, 4096, 16384]  # Possible M values

    best\_M = None

    best\_a = None

    for M in possible\_Ms:

        a = np.log2(M) \* W / R - 1

        if 0 < a < 1:

            best\_M = M

            best\_a = a

            break

    return best\_a, best\_M

# Input widgets for frequencies and data rate

f1\_input = widgets.FloatText(description='F1 (MHz):', value=6.75)

f2\_input = widgets.FloatText(description='F2 (MHz):', value=9.25)

R\_input = widgets.FloatText(description='R (Mbps):', value=12)

# Function to update and display the result

def update\_result(f1, f2, R):

    # Start timer and display loading animation

    loader\_html3.value = loading

    start\_time = time.time()

    a, M = calculate\_rolloff(f1, f2, R)

    # Update elapsed time and loading animation

    elapsed\_time = time.time() - start\_time

    timer\_html3.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html3.value = done

    # Display the result

    if M is not None:

        result\_html.value = f"Result: α = {a:.4f}, Recommended M = {M}"

    else:

        result\_html.value = "No suitable M found within the given constraints."

# Widget to display the result

result\_html = widgets.HTML(value="Result: α = 0.2500, Recommended M = 64")

# Group the loader and timer widgets vertically

loader\_timer\_box = widgets.VBox([loader\_html3, timer\_html3], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Arrange input widgets

input\_widgets = widgets.VBox([f1\_input, f2\_input, R\_input], layout=widgets.Layout(width='auto'))

# Combine inputs and loader/timer into a horizontal box

ui = widgets.HBox([input\_widgets, loader\_timer\_box], layout=widgets.Layout(align\_items='center'))

# Create interactive output

out = widgets.interactive\_output(update\_result, {'f1': f1\_input, 'f2': f2\_input, 'R': R\_input})

# Clear previous output and display the UI

clear\_output(wait=True)

display(ui, result\_html)

# Loading animation and completion checkmark HTML code

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

# Create HTML widgets for loading animation and elapsed time

loader\_html4 = widgets.HTML(

  value=loading

)

timer\_html4 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# HTML widget for displaying warnings

warning\_html = widgets.HTML(

    value=""

)

def rootRaisedCosine(nsamp, roll\_off, delay):

    """

    Generate a root raised cosine filter.

    Parameters:

    - nsamp: Number of samples per symbol

    - roll\_off: Roll-off factor

    - delay: Filter delay in symbol periods

    Returns:

    - h: Filter coefficients

    """

    # Calculate necessary frequencies and times

    F0 = 0.5 / nsamp

    Br = 1

    Fs = Br \* nsamp

    Td = 1 / Br

    Ts = 1 / Fs

    F1 = F0 \* (1 - roll\_off)

    F2 = F0 \* (1 + roll\_off)

    filter\_order = 2 \* nsamp \* delay

    t = np.arange(0, filter\_order, Td)

    h = []

    for i in range(len(t)):

        t\_shifted = t[i] - filter\_order / 2

        if t\_shifted == 0:

            h.append(np.sqrt(2 \* F0) \*(1 + roll\_off \* ((4 / np.pi) - 1)))

        elif t\_shifted == 1 / 8 / roll\_off / F0 or t\_shifted == - 1 / 8 / roll\_off / F0 :

            h.append((roll\_off \* np.sqrt(F0)) \* ((1 + 2 / np.pi) \* np.sin(np.pi / 4 / roll\_off) + (1 - 2 / np.pi) \* np.cos(np.pi / 4 / roll\_off)))

        else:

            factor1 = np.sqrt(2 \* F0) / (1 - 64 \* roll\_off \* roll\_off \* F0 \* F0 \* t\_shifted \* t\_shifted)

            factor2 = np.sin(2 \* np.pi \* F1 \* t\_shifted) / (2 \* np.pi \* F0 \* t\_shifted)

            factor3 = (4 \* roll\_off / np.pi) \* np.cos(2 \* np.pi \* F2 \* t\_shifted)

            h.append(factor1 \* (factor2 + factor3))

    return h

def ber\_qam(EbNo, M, roll\_off, F1, F2, Br):

    """

    Calculate the Bit Error Rate (BER) for QAM modulation.

    Parameters:

    - EbNo: Energy per bit to noise power spectral density ratio in dB

    - M: Modulation order

    - roll\_off: Roll-off factor

    - F1: Lower frequency in MHz

    - F2: Upper frequency in MHz

    - Br: Bit rate in Mbps

    Returns:

    - BER: Bit Error Rate

    """

    # Convert units from MHz and Mbps to Hz and bps

    F1 = F1 \* 1e6

    F2 = F2 \* 1e6

    Br = Br \* 1e6

    W = F2 - F1         # Bandwidth in Hz

    fc = F1 + W / 2     # Carrier frequency

    nsamp = int(np.ceil(2 \* F2 / Br)) +7  # Number of samples per symbol

    a = 0.25    # Roll-off factor

    L = int(np.sqrt(M))

    l = np.log2(L)

    k = 2 \* l

    Nsymb = 10000

    SNR = EbNo - 10 \* np.log10(nsamp / k / 2)  # σε db

    core = [1+1j, 1-1j, -1+1j, -1-1j]

    mapping = core[:]

    if l > 1:

        for j in range(1, int(l)):

            mapping = list(map(lambda x: x + j \* 2 \* core[0], mapping))

            conj\_arr = np.conj(mapping)

            mapping = mapping + conj\_arr.tolist()

            conj\_arr = -np.conj(mapping)

            mapping = mapping + conj\_arr.tolist()

    # Generate random sequence

    x = np.floor(2 \* np.random.rand(int(k \* Nsymb), 1))

    x\_temp = np.reshape(x, (int(len(x) / (k)), int(k)))

    xsym = []

    # Split the list into sublists and put the contents of each sublist

    # into a string so that with the int() command it is converted from binary to decimal

    for i in range(len(x\_temp)):

        my\_str = ''

        y = x\_temp[i]

        for j in range(int(np.log2(M))):

            my\_str = my\_str + str(int(y[j]))

        a = int(my\_str, 2)

        xsym = xsym + [a]

    y = []

    for n in range(len(xsym)):

        y = y + [mapping[xsym[n]]]

    # Generate shaping filter

    delay = 10

    filtorder = delay \* nsamp \* 2

    shaping\_filter = rootRaisedCosine(nsamp, roll\_off, delay)

    # Transmit signal

    ytx = upfirdn([1], y, nsamp)  # upsample

    ytx = np.convolve(ytx, shaping\_filter)

    m = np.arange(1, len(ytx) + 1)

    s = np.real(np.multiply(ytx, np.exp(1j \* 2 \* np.pi \* fc \* m / nsamp)))

    s\_matrix = np.matrix(s)  # transpose

    s\_matrix = s\_matrix.getH()

    s\_list = s\_matrix.tolist()

    Ps = 10 \* np.log10(np.matmul(s, s\_list) / len(s))  # Power of complex signal in dB

    Pn = Ps - SNR

    n = np.sqrt(10\*\*(Pn / 10)) \* np.random.randn(1, len(ytx))

    snoisy = s + n

    # receiver

    yrx = 2 \* np.multiply(snoisy, np.exp(-1j \* 2 \* np.pi \* fc \* m / nsamp))

    yrx = yrx[0, :]

    yrx = np.convolve(yrx, shaping\_filter)

    yrx = yrx[::nsamp]  # downsample

    yrx = yrx[2 \* delay + 0:len(yrx) - 2 \* delay]

    yi = yrx.copy()

    yq = np.imag(yi)

    yi = np.real(yi)

    xrx = []

    q = np.arange(-L + 1, L, 2)

    for n in range(len(yrx)):

        differences = np.abs(q - yi[n])  # Array with the differences of the signal from the levels

        m = min(differences)

        [index], = np.where(differences == m)

        yi[n] = q[index]

        differences = np.abs(q - yq[n])  # Array with the differences of the signal from the levels

        m = min(differences)

        [index], = np.where(differences == m)

        yq[n] = q[index]

    error = 0

    for i in range(len(yrx)):

        if y[i] != yi[i] + yq[i] \* 1j:

            error += 1

    return error / len(x)

def plot\_ber\_qam(M, roll\_off, F1, F2, Br):

    """

    Plot the BER curve for QAM modulation.

    Parameters:

    - M: Modulation order

    - roll\_off: Roll-off factor

    - F1: Lower frequency in MHz

    - F2: Upper frequency in MHz

    - Br: Bit rate in Mbps

    """

    # Validate input frequencies

    if F1 >= F2:

        warning\_html.value = "<div style='color: red; font-size: 16px;'>Warning: F1 should be less than F2.</div>"

        return

    warning\_html.value = ""  # Clear any previous warnings

    # Start the timer

    loader\_html4.value = loading

    start\_time = time.time()

    ber\_exp = []

    ber\_th = []

    L = int(np.sqrt(M))

    for i in range(1, 15):

        ber\_exp.append(ber\_qam(i, M, roll\_off, F1, F2, Br))

        ber\_th.append(((L - 1) /(L\*np.log2(L)) \* scipy.special.erfc(np.sqrt(3 \* np.log2(L) / (L \* L - 1) \* 10\*\*(i/10)))))

    plt.figure(figsize=(10, 8))

    plt.semilogy(range(1, 15), ber\_th)  # Plot theoretical BER as a line

    plt.semilogy(range(1, 15), ber\_exp, 'o')  # Plot experimental BER as points

    plt.legend(['Theoretical', 'Simulation'])

    plt.xlabel('Eb/N0(db)')

    plt.ylabel('Bit Error Probability')

    plt.title(f'BER curve for {M}-QAM')

    plt.grid(which='both')

    plt.show()

    # Stop the timer and update the timer HTML

    elapsed\_time = time.time() - start\_time

    timer\_html4.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    # Update the loading animation to done

    loader\_html4.value = done

# Define QAM options

qam\_options = {'4-QAM': 4, '16-QAM': 16, '64-QAM': 64}

qam\_selector = widgets.Dropdown(options=qam\_options, value=16, description='QAM Type:')

# Define additional input boxes for roll-off, F1, F2, and Br

roll\_off\_input = widgets.FloatText(value=0.25, description='Roll-off:')

F1\_input = widgets.FloatText(value=6.75, description='F1: (MHz)')

F2\_input = widgets.FloatText(value=9.25, description='F2: (MHz)')

Br\_input = widgets.FloatText(value=10, description='Br: (Mbps)')

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox([loader\_html4, timer\_html4], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Create a VBox for the input widgets (similar to the first code snippet)

input\_widgets = widgets.VBox([qam\_selector, roll\_off\_input, F1\_input, F2\_input, Br\_input, warning\_html], layout=widgets.Layout(width='auto'))

# Create an HBox to combine inputs and loader timer box, with the same layout style

ui = widgets.HBox([input\_widgets, loader\_timer\_box], layout=widgets.Layout(align\_items='center'))

# Create the interactive output for the plot\_ber\_qam function

out = widgets.interactive\_output(plot\_ber\_qam, {'M': qam\_selector, 'roll\_off': roll\_off\_input, 'F1': F1\_input, 'F2': F2\_input, 'Br': Br\_input})

# Display the UI and output

clear\_output(wait=True)  # Clear the previous output

display(ui, out)

Μείωση Τάξης Συστήματος QAM για Βελτίωση Πιθανότητας Σφάλματος Bit και Ανάλυση Ρυθμού Μετάδοσης

# Loading animation and done checkmark HTML code

# These HTML snippets are used to display a loading spinner and a completion checkmark in the UI.

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Greenish-blue border on top \*/

                                     border-radius: 50%;             /\* Make it a circle \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spin animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Rotate full circle \*/

    }

    </style>

    """

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark -->

    </div>

    """

# Create HTML widgets for the loading animation and timer display

loader\_html5 = widgets.HTML(value=loading)

timer\_html5 = widgets.HTML(value="Elapsed time: - seconds")

def rtrapezium(nsamp, rolloff, delay):

    """

    Generate a root-raised cosine (RRC) filter using the trapezium method.

    Parameters:

    - nsamp (int): Number of samples per symbol (oversampling factor)

    - rolloff (float): Roll-off factor (0 < rolloff <= 1)

    - delay (int): Filter delay in symbol periods

    Returns:

    - rrc (numpy array): Filter coefficients

    """

    T = 1  # Symbol period

    t = np.arange(-delay\*T, (delay\*T) + 1/nsamp, 1/nsamp)  # Time vector

    rrc = np.zeros\_like(t)  # Initialize filter coefficients

    # Calculate filter coefficients

    for i, ti in enumerate(t):

        if ti == 0.0:

            rrc[i] = 1.0 - rolloff + 4 \* rolloff / np.pi

        elif abs(ti) == T / (4 \* rolloff):

            rrc[i] = (rolloff / np.sqrt(2)) \* (((1 + 2/np.pi) \* np.sin(np.pi / (4 \* rolloff))) +

                                               ((1 - 2/np.pi) \* np.cos(np.pi / (4 \* rolloff))))

        else:

            numerator = (np.sin(np.pi \* ti \* (1 - rolloff) / T) +

                         4 \* rolloff \* ti \* np.cos(np.pi \* ti \* (1 + rolloff) / T) / T)

            denominator = (np.pi \* ti \* (1 - (4 \* rolloff \* ti / T) \*\* 2))

            rrc[i] = numerator / denominator

    # Normalize filter coefficients

    rrc = rrc / np.sqrt(np.sum(rrc\*\*2))

    return rrc

def run\_simulation(f1, f2, qam\_type):

    """

    Simulate the transmission and reception of a QAM signal and plot the power spectral density.

    Parameters:

    - f1 (float): Lower cutoff frequency in MHz

    - f2 (float): Upper cutoff frequency in MHz

    - qam\_type (int): Modulation order (e.g., 16 for 16-QAM)

    """

    # Start the timer and display loading animation

    loader\_html5.value = loading

    start\_time = time.time()

    # Parameters

    k = int(np.log2(qam\_type))   # Bits per symbol

    M = 2\*\*k                     # Modulation order

    Nsymb = 30000                # Number of symbols

    pulse\_type = 1               # Pulse shaping filter type (1 for RRC)

    nsamp = 32                   # Oversampling factor

    fc = (f1 + f2) / 2           # Carrier frequency (MHz)

    bandwidth = f2 - f1          # Signal bandwidth (MHz)

    rolloff = bandwidth / (2 \* fc)  # Roll-off factor

    EbNo = 10                    # Energy per bit to noise power spectral density ratio in dB

    SNR = EbNo - 10 \* np.log10(nsamp / k / 2)  # Signal-to-noise ratio per sample

    # Phase and mapping initialization

    ph1 = np.pi / 4

    theta = np.array([ph1, -ph1, np.pi - ph1, -np.pi + ph1])

    mapping = np.exp(1j \* theta)

    if k > 2:

        # Generate PSK constellation for higher-order modulation

        for j in range(3, k + 1):

            theta = theta / 2

            mapping = np.exp(1j \* theta)

            mapping = np.concatenate([mapping, -np.conj(mapping)])

            theta = np.angle(mapping)

    # Transmitter

    x = np.random.randint(0, 2, k \* Nsymb)  # Random binary sequence

    xsym = x.reshape(-1, k)

    xsym = xsym.dot(2\*\*np.arange(xsym.shape[-1])[::-1])  # Convert bits to decimal symbols

    y = mapping[xsym]  # Map symbols to constellation points

    # Shaping filter definition

    if pulse\_type == 1:  # Nyquist pulse shaping with RRC filter

        delay = 8  # Group delay in symbol periods

        shaping\_filter = rtrapezium(nsamp, rolloff, delay)

    else:  # Rectangular pulse

        delay = 0.5

        shaping\_filter = np.ones(nsamp) / np.sqrt(nsamp)  # Normalize filter

    # Transmitted signal

    ytx = upfirdn([1], y, nsamp)  # Upsample the signal

    ytx = convolve(ytx, shaping\_filter, mode='same')  # Apply shaping filter

    # Quadrature modulation

    m = np.arange(len(ytx))

    s = np.real(ytx \* np.exp(1j \* 2 \* np.pi \* fc \* m / nsamp))  # Modulate to carrier frequency

    # Adding white Gaussian noise

    Ps = 10 \* np.log10(np.mean(s\*\*2))  # Signal power in dB

    Pn = Ps - SNR                      # Noise power in dB

    n = np.sqrt(10\*\*(Pn / 10)) \* np.random.randn(len(ytx))  # Generate noise

    snoisy = s + n  # Noisy bandpass signal

    # Receiver

    yrx = 2 \* snoisy \* np.exp(-1j \* 2 \* np.pi \* fc \* m / nsamp)  # Demodulate

    yrx = convolve(yrx, shaping\_filter, mode='same')             # Apply matched filter

    # Spectrum plot of received signal

    f, Pxx\_den = welch(np.real(s), fs=nsamp, nperseg=1024)

    Pxx\_den = 10 \* np.log10(Pxx\_den)  # Convert power spectral density to dB

    plt.figure(figsize=(10, 8))

    plt.plot(f, Pxx\_den, 'r')

    plt.title('Welch Power Spectral Density Estimate')

    plt.xlabel('Frequency (Hz)')

    plt.ylabel('Power Spectral Density (dB/Hz)')

    plt.xlim(0, max(f1, f2) + 5)

    plt.grid()

    plt.show()

    # Stop the timer and update the loading animation

    elapsed\_time = time.time() - start\_time

    timer\_html5.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html5.value = done

# Widgets for input parameters

f1\_widget = widgets.FloatText(value=6.75, description='f1 (MHz):')

f2\_widget = widgets.FloatText(value=9.25, description='f2 (MHz):')

qam\_widget = widgets.Dropdown(options=[4, 16, 64], value=16, description='QAM Type:')

# Create the interactive output for the run\_simulation function

out = widgets.interactive\_output(run\_simulation, {'f1': f1\_widget, 'f2': f2\_widget, 'qam\_type': qam\_widget})

# Group the loader and timer together vertically

loader\_timer\_box = widgets.VBox([loader\_html5, timer\_html5], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Arrange input widgets in a vertical box

input\_widgets = widgets.VBox([qam\_widget, f1\_widget, f2\_widget], layout=widgets.Layout(width='auto'))

# Combine input widgets and loader/timer into a horizontal box

ui = widgets.HBox([input\_widgets, loader\_timer\_box], layout=widgets.Layout(align\_items='center'))

# Display the UI and output

display(ui, out)

Αύξηση Ρυθμού Μετάδοσης με Μείωση του Roll-Off του Φίλτρου Nyquist κατά 50%

# Loading animation and done checkmark HTML code

# These HTML snippets are used to display a loading spinner and a completion checkmark in the UI.

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Greenish-blue border on top \*/

                                     border-radius: 50%;             /\* Make it a circle \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spin animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Rotate full circle \*/

    }

    </style>

    """

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark -->

    </div>

    """

loader\_html6 = widgets.HTML(value=loading)

timer\_html6 = widgets.HTML(value="Elapsed time: - seconds")

# Function to calculate the maximum achievable transmission rate R' and the percentage increase

def calculate\_R\_and\_percentage\_increase(M, a, R):

    """

    Calculate the maximum achievable rate R' and the percentage increase over the current rate R.

    Parameters:

    - M (float): Modulation order

    - a (float): Roll-off factor

    - R (float): Current data rate in Mbps

    Returns:

    - R\_prime\_mbps (float): Maximum achievable rate in Mbps

    - percentage\_increase (float): Percentage increase over the current rate

    """

    W = 2.5 \* 1e6  # Fixed bandwidth in Hz

    log2M = np.log2(M)

    R\_prime = (log2M \* W) / (1 + a)  # Maximum achievable rate in bps

    R\_prime\_mbps = R\_prime / 1e6     # Convert to Mbps

    percentage\_increase = ((R\_prime\_mbps - R) / R) \* 100  # Percentage increase

    return R\_prime\_mbps, percentage\_increase

# Input widgets for R, M, and a

R\_input = widgets.FloatText(description='R (Mbps):', value=8.0)

M\_input = widgets.FloatText(description='M:', value=16)

a\_input = widgets.FloatText(description='α (Roll-off):', value=0.125)

# Output widget for displaying results

output\_vals = widgets.Output()

# Function to update the result when any input changes

def update\_result(change=None):

    # Start the timer and display loading animation

    loader\_html6.value = loading

    start\_time = time.time()

    with output\_vals:

        output\_vals.clear\_output()

        R = R\_input.value

        M = M\_input.value

        a = a\_input.value

        R\_prime\_mbps, percentage\_increase = calculate\_R\_and\_percentage\_increase(M, a, R)

        print(f"Maximum Achievable Rate (R') = {R\_prime\_mbps:.3f} Mbps")

        print(f"Percentage Increase = {percentage\_increase:.2f}%")

        # Stop the timer and update the loading animation

        elapsed\_time = time.time() - start\_time

        timer\_html6.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

        loader\_html6.value = done

# Attach observers to input widgets to trigger result update

R\_input.observe(update\_result, names='value')

M\_input.observe(update\_result, names='value')

a\_input.observe(update\_result, names='value')

# Group the loader and timer together vertically

loader\_timer\_box = widgets.VBox([loader\_html6, timer\_html6], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Arrange input widgets in a vertical box

input\_widgets = VBox([R\_input, M\_input, a\_input], layout=Layout(width='auto'))

# Combine input widgets and loader/timer into a horizontal box

inputs\_and\_loader = HBox([input\_widgets, loader\_timer\_box], layout=Layout(align\_items='center'))

# Create a vertical box that includes both inputs and output

ui = VBox([inputs\_and\_loader, output\_vals])

# Clear previous output and display the UI

clear\_output(wait=True)

display(ui)

# Initial calculation to display results

update\_result()

Σύγκριση Συστήματος PSK και QAM ως προς BER και Εύρος Ζώνης με Κωδικοποίηση Gray

# HTML for loading animation and completion checkmark

# These HTML snippets are used to display a loading spinner and a completion checkmark in the UI.

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Greenish-blue border on top \*/

                                     border-radius: 50%;             /\* Make it a circle \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spin animation \*/'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start at 0 degrees \*/

        100% { transform: rotate(360deg); } /\* Rotate full circle \*/

    }

    </style>

    """

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Unicode checkmark -->

    </div>

    """

loader\_html7 = widgets.HTML(value=loading)

timer\_html7 = widgets.HTML(value="Elapsed time: - seconds")

# Define the root raised cosine filter function

def rootRaisedCosine1(nsamp, roll\_off, delay):

    """

    Generate a root raised cosine (RRC) filter.

    Parameters:

    - nsamp (int): Number of samples per symbol

    - roll\_off (float): Roll-off factor

    - delay (int): Filter delay in symbol periods

    Returns:

    - h (numpy array): Filter coefficients

    """

    t = np.arange(-delay, delay + 1 / nsamp, 1 / nsamp)

    h = np.zeros(len(t))

    for i in range(len(t)):

        if t[i] == 0.0:

            h[i] = 1.0 - roll\_off + 4 \* roll\_off / np.pi

        elif roll\_off != 0 and t[i] == 1 / (4 \* roll\_off):

            h[i] = roll\_off / np.sqrt(2) \* ((1 + 2 / np.pi) \* np.sin(np.pi / (4 \* roll\_off)) + (1 - 2 / np.pi) \* np.cos(np.pi / (4 \* roll\_off)))

        elif roll\_off != 0 and t[i] == -1 / (4 \* roll\_off):

            h[i] = roll\_off / np.sqrt(2) \* ((1 + 2 / np.pi) \* np.sin(np.pi / (4 \* roll\_off)) + (1 - 2 / np.pi) \* np.cos(np.pi / (4 \* roll\_off)))

        else:

            h[i] = (np.sin(np.pi \* t[i] \* (1 - roll\_off)) + 4 \* roll\_off \* t[i] \* np.cos(np.pi \* t[i] \* (1 + roll\_off))) / (np.pi \* t[i] \* (1 - (4 \* roll\_off \* t[i]) \*\* 2))

    return h

# Define the BER computation functions for PSK modulation

def compute\_ber\_psk(EbNo\_dB, M1):

    """

    Compute the theoretical Bit Error Rate (BER) for M-PSK modulation.

    Parameters:

    - EbNo\_dB (float): Eb/N0 ratio in dB

    - M1 (int): Modulation order (e.g., 2 for BPSK, 4 for QPSK)

    Returns:

    - BER (float): Bit Error Rate

    """

    EbNo\_linear = 10\*\*(EbNo\_dB / 10)

    if M1 == 2: # BPSK

        return 0.5 \* scipy.special.erfc(np.sqrt(EbNo\_linear))

    else:       # M-PSK

        k = np.log2(M1)

        return (1/4\*k) \* scipy.special.erfc(np.sqrt(EbNo\_linear \* k) \* np.sin(np.pi / M1))

def ber\_psk\_simulation(EbNo\_dB, M1):

    """

    Simulate the Bit Error Rate (BER) for M-PSK modulation over an AWGN channel.

    Parameters:

    - EbNo\_dB (float): Eb/N0 ratio in dB

    - M1 (int): Modulation order

    Returns:

    - ber (float): Bit Error Rate from simulation

    """

    Nsymb = 30000   # Number of symbols

    nsamp = 16      # Samples per symbol

    fc = 4          # Carrier frequency

    rolloff = 0.25

    delay = 10

    SNR\_dB = EbNo\_dB - 10 \* np.log10(nsamp / np.log2(M1))

    shaping\_filter = rootRaisedCosine1(nsamp, rolloff, delay)

    filtorder = delay \* nsamp \* 2

    # Generate random symbol sequence

    bits1 = np.random.randint(0, M1, Nsymb)

    # Map symbols to PSK constellation points

    symbols = np.exp(1j \* (2 \* np.pi \* bits1 / M1))

    # Upsample and filter the signal

    ytx1 = upfirdn([1], symbols, nsamp)

    ytx1 = np.convolve(ytx1, shaping\_filter, mode='same')

    m1 = np.arange(len(ytx1))

    s1 = np.real(ytx1 \* np.exp(1j \* 2 \* np.pi \* fc \* m1 / nsamp))

    # Calculate signal and noise power

    Ps = np.mean(np.abs(s1)\*\*2)

    SNR\_linear = 10\*\*(SNR\_dB / 10)

    Pn = Ps / SNR\_linear

    # Generate AWGN noise

    noise = np.sqrt(Pn / 4) \* (np.random.randn(len(s1)) + 1j \* np.random.randn(len(s1)))

    snoisy = s1 + noise

    # Receiver

    yrx1 = snoisy \* np.exp(-1j \* 2 \* np.pi \* fc \* m1 / nsamp)

    yrx1 = np.convolve(yrx1, shaping\_filter, mode='same')

    yrx1 = yrx1[::nsamp]

    # Demodulate symbols

    detected\_bits1 = np.angle(yrx1) \* M1 / (2 \* np.pi)

    detected\_bits1 = np.round(detected\_bits1) % M1

    # Calculate BER

    bit\_errors1 = np.sum(bits1 != detected\_bits1)

    ber1 = bit\_errors1 / len(bits1)

    return ber1

def plot\_ber\_psk(M):

    """

    Plot the BER curve for M-PSK modulation.

    Parameters:

    - M (int): Modulation order (e.g., 2 for BPSK, 4 for QPSK)

    """

    # Start the timer and display loading animation

    loader\_html7.value = loading

    start\_time = time.time()

    ber\_exp = []  # Experimental BER

    ber\_th = []   # Theoretical BER

    for i in range(1, 18):

        ber\_exp.append(ber\_psk\_simulation(i, M))

        ber\_th.append(compute\_ber\_psk(i, M))

    # Plot BER curves

    plt.figure(figsize=(10, 8))

    plt.semilogy(range(1, 18), ber\_th)  # Plot theoretical BER as a line

    plt.semilogy(range(1, 18), ber\_exp, 'o')  # Plot experimental BER as points

    plt.legend(['Theoretical', 'Simulation'])

    plt.xlabel('Eb/N0 (dB)')

    plt.ylabel('Bit Error Probability')

    plt.title(f'BER curve for {M}-PSK')

    plt.grid(which='both')

    plt.show()

    # Stop the timer and update the timer HTML

    elapsed\_time = time.time() - start\_time

    timer\_html7.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    # Update the loading animation to done

    loader\_html7.value = done

# Define PSK options for the dropdown menu

psk\_options = {'BPSK': 2, 'QPSK': 4, '8-PSK': 8}

psk\_selector = widgets.Dropdown(options=psk\_options, value=4, description='PSK Type:')

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox([loader\_html7, timer\_html7], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Create a VBox for the input widgets (similar to the first code snippet)

input\_widgets = widgets.VBox([psk\_selector], layout=widgets.Layout(width='auto'))

# Create an HBox to combine inputs and loader timer box, with the same layout style

ui = widgets.HBox([input\_widgets, loader\_timer\_box], layout=widgets.Layout(align\_items='center'))

# Create the interactive output for the plot\_ber\_psk function

out = widgets.interactive\_output(plot\_ber\_psk, {'M': psk\_selector})

# Display the UI and output

clear\_output(wait=True)  # Clear the previous output

display(ui, out)

### Lab Exercise 6: Διαμόρφωση FSK και MSK

Εκτέλεση Κώδικα 6.1 για M=16 και Επιβεβαίωση Αριθμού Λαθών για Διαφορετικές Τιμές Eb/No

# Loading animation and completion checkmark HTML code for visual feedback

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'>

        </div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation \*/

        100% { transform: rotate(360deg); } /\* End rotation \*/

    }

    </style>

    """

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Checkmark symbol -->

    </div>

    """

# Create HTML widgets to display the loading animation and elapsed time

loader\_html1 = widgets.HTML(value=loading)

timer\_html1 = widgets.HTML(value="Elapsed time: - seconds")

# Parameters for FSK modulation

bps = 4           # Bits per symbol (4 bits per symbol for 16-FSK)

Nsymb = 1000      # Number of symbols to simulate

ns = 80           # Number of samples per symbol (oversampling factor)

# Derived parameters

M = 2 \*\* bps      # Number of different symbols (M-ary FSK)

BR = 1            # Baud Rate (symbols per second)

fc = 2 \* M \* BR   # Carrier frequency for FSK modulation

# Further derived parameters

nb = bps \* Nsymb  # Total number of bits to simulate

T = 1 / BR        # Symbol duration

Ts = T / ns       # Sampling period (time between samples)

# Generate M frequencies for coherent FSK modulation

f = fc + (BR / 2) \* (np.arange(1, M + 1) - (M + 1) / 2)

# Calculate the maximum frequency used

fmax = np.max(f)

# Recalculate 'ns' to ensure the sampling frequency 'Fs' satisfies the Nyquist criterion

Fs = 2 \* fmax                    # Sampling frequency must be greater than twice the max frequency

ns = int(np.ceil(Fs / BR)) + 10  # Adjust 'ns' accordingly and add some margin

# Recalculate the sampling period with the new 'ns'

Ts = T / ns

# Generate random input data bits

y = np.random.randint(0, 2, nb)   # Random bits (0 or 1)

x = y.reshape((Nsymb, bps))       # Reshape bits into symbols (each symbol has 'bps' bits)

# Time vectors for symbol intervals and oversampling

t = np.arange(0, len(x) \* T, T)   # Time vector on the symbol grid

tks = np.arange(0, T, Ts)         # Time vector for oversampling within one symbol period

# Generate FSK signal

s = []                             # Initialize signal list

A = np.sqrt(2 / T / ns)            # Amplitude to normalize power

for k in range(len(x)):

    # Convert bits to symbol index

    symbol\_bits = ''.join(map(str, x[k]))

    symbol\_index = int(symbol\_bits, 2)

    fk = f[symbol\_index]           # Select frequency for the symbol

    tk = (k \* T) + tks             # Time vector for the current symbol

    s.append(np.sin(2 \* np.pi \* fk \* tk))  # Generate sine wave for the symbol

s = np.concatenate(s)              # Concatenate all symbol signals

def calculate\_errors(EbNo\_range):

    """

    Calculate the number of errors over a range of Eb/No values.

    Parameters:

    - EbNo\_range: Tuple containing the start and end Eb/No values in dB

    """

    # Start timer and display loading animation

    loader\_html1.value = loading

    start\_time = time.time()

    clear\_output(wait=True)  # Clear previous output

    EbNo\_values = list(range(EbNo\_range[0], EbNo\_range[1] + 1))  # List of Eb/No values

    errors\_list = []  # List to store the number of errors at each Eb/No

    for EbNo in EbNo\_values:

        # Calculate Signal-to-Noise Ratio (SNR) in dB

        SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)

        # Add AWGN noise to the FSK signal

        noisy\_signal = awgn(s, SNR)

        # FSK receiver implementation

        xr = []  # List to store received symbols

        for k in range(len(noisy\_signal) // ns):

            tk = (k \* T) + tks                      # Time vector for the current symbol

            sk = noisy\_signal[k \* ns:(k + 1) \* ns]  # Extract the symbol from the noisy signal

            smi = []                                # List to store correlation values

            for fi in f:

                si = np.sin(2 \* np.pi \* fi \* tk)    # Reference sine wave for frequency 'fi'

                smi.append(np.sum(sk \* si))         # Correlate received signal with reference

            j = np.argmax(smi)                      # Find the index with the maximum correlation

            # Convert index to bits and append to received symbols

            xr.append([int(bit) for bit in bin(j)[2:].zfill(bps)])

        xr = np.array(xr).reshape((Nsymb, bps))     # Reshape received symbols

        # Count the number of bit errors

        errors = np.sum(x != xr)

        errors\_list.append(errors)

    # Plot the number of errors versus Eb/No

    plt.figure(figsize=(10, 6))

    plt.plot(EbNo\_values, errors\_list, marker='o', linestyle='-', markersize=8)

    plt.xlabel('Eb/No (dB)')

    plt.ylabel('Number of Errors')

    plt.title('Number of Errors vs. Eb/No')

    plt.grid(True)

    plt.show()

    # Display elapsed time and completion checkmark

    elapsed\_time = time.time() - start\_time

    timer\_html1.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html1.value = done

# Create a range slider widget for Eb/No values

EbNo\_slider = IntRangeSlider(

    value=[0, 20],          # Default range from 0 to 20 dB

    min=0,                  # Minimum value

    max=20,                 # Maximum value

    step=1,                 # Step size

    description='EbNo (dB):',

    continuous\_update=False,

    layout=Layout(width='99%')

)

# Group the loader and timer widgets

loader\_timer\_box = widgets.VBox(

    [loader\_html1, timer\_html1],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Create a VBox for input widgets

input\_widgets = widgets.VBox(

    [EbNo\_slider],

    layout=widgets.Layout(flex='1 1 auto', width='auto')

)

# Combine input widgets and loader/timer into an HBox

ui = widgets.HBox(

    [input\_widgets, loader\_timer\_box],

    layout=widgets.Layout(align\_items='center')

)

# Create the interactive output

out = widgets.interactive\_output(calculate\_errors, {'EbNo\_range': EbNo\_slider})

# Display the UI and output

clear\_output(wait=True)  # Clear previous output

display(ui, out)

Εξομοίωση Συστήματος 32-FSK και Σύγκριση Καμπυλών Pb-Eb/No για Σύμφωνη και Ασύμφωνη Αποδιαμόρφωση

# Loading animation and completion checkmark HTML code

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'>

        </div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation \*/

        100% { transform: rotate(360deg); } /\* End rotation \*/

    }

    </style>

    """

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Checkmark symbol -->

    </div>

    """

loader\_html2 = widgets.HTML(value=loading)

timer\_html2 = widgets.HTML(value="Elapsed time: - seconds")

def fsk\_errors(bps, Nsymb, ns, EbNo):

    """

    Simulate coherent FSK and calculate the number of bit errors.

    Parameters:

    - bps: Bits per symbol

    - Nsymb: Number of symbols

    - ns: Samples per symbol

    - EbNo: Energy per bit to noise power spectral density ratio in dB

    Returns:

    - errors: Number of bit errors

    """

    # Input parameters

    M = 2 \*\* bps              # Modulation order

    BR = 1                    # Baud Rate

    fc = 2 \* M \* BR           # Carrier frequency

    # Derived parameters

    nb = bps \* Nsymb          # Total number of bits

    T = 1 / BR                # Symbol duration

    Ts = T / ns               # Sampling period

    # Generate frequencies for coherent FSK

    f = fc + (BR / 2) \* (np.arange(1, M + 1) - (M + 1) / 2)

    # Calculate the maximum frequency and adjust 'ns' accordingly

    fmax = np.max(f)

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    Ts = T / ns

    # Calculate SNR in dB

    SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)

    # Generate random input bits

    y = np.random.randint(0, 2, nb)

    x = y.reshape((Nsymb, bps))

    # Time vectors

    t = np.arange(0, len(x) \* T, T)

    tks = np.arange(0, T, Ts)

    # Generate FSK signal

    s = []

    A = np.sqrt(2 / T / ns)

    for k in range(len(x)):

        symbol\_bits = ''.join(map(str, x[k]))

        symbol\_index = int(symbol\_bits, 2)

        fk = f[symbol\_index]

        tk = (k \* T) + tks

        s.append(np.sin(2 \* np.pi \* fk \* tk))

    s = np.concatenate(s)

    # Add AWGN noise to the signal

    s = awgn(s, SNR)

    # FSK receiver

    xr = []

    for k in range(len(s) // ns):

        tk = (k \* T) + tks

        sk = s[k \* ns:(k + 1) \* ns]

        smi = []

        for fi in f:

            si = np.sin(2 \* np.pi \* fi \* tk)

            smi.append(np.sum(sk \* si))

        j = np.argmax(smi)

        xr.append([int(bit) for bit in bin(j)[2:].zfill(bps)])

    xr = np.array(xr).reshape((Nsymb, bps))

    # Count errors

    errors = np.sum(x != xr)

    return errors

def fsk\_errors\_non\_coh(bps, Nsymb, ns, EbNo):

    """

    Simulate non-coherent FSK and calculate the number of bit errors.

    Parameters:

    - bps: Bits per symbol

    - Nsymb: Number of symbols

    - ns: Samples per symbol

    - EbNo: Energy per bit to noise power spectral density ratio in dB

    Returns:

    - errors: Number of bit errors

    """

    M = 2 \*\* bps            # Modulation order

    BR = 1                  # Baud Rate

    fc = 2 \* M \* BR         # Carrier frequency

    # Derived parameters

    nb = bps \* Nsymb

    T = 1 / BR

    Ts = T / ns

    # Generate frequencies for non-coherent FSK

    f = fc + BR \* (np.arange(1, M + 1) - (M + 1) / 2)

    # Adjust 'ns' based on maximum frequency

    fmax = np.max(f)

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    Ts = T / ns

    # Calculate SNR in dB

    SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)

    # Generate random input bits

    y = np.random.randint(0, 2, nb)

    x = y.reshape((-1, bps))

    t = np.arange(0, len(x) \* T, T)

    tks = np.arange(0, T, Ts)

    # Generate FSK signal

    s = []

    A = np.sqrt(2 / T / ns)

    for k in range(len(x)):

        symbol\_bits = ''.join(map(str, x[k]))

        symbol\_index = int(symbol\_bits, 2)

        fk = f[symbol\_index]

        tk = (k \* T) + tks

        s = np.concatenate((s, np.sin(2 \* np.pi \* fk \* tk)))

    # Add AWGN noise

    s = awgn(s, SNR)

    # Non-coherent FSK receiver

    xr = []

    for k in range(len(s) // ns):

        tk = (k \* T) + tks

        sk = s[k \* ns:(k + 1) \* ns]

        sm = []

        for i in range(M):

            si = np.sin(2 \* np.pi \* f[i] \* tk)

            sq = np.cos(2 \* np.pi \* f[i] \* tk)

            smi = np.sum(sk \* si)

            smq = np.sum(sk \* sq)

            sm.append(np.sqrt(smi \*\* 2 + smq \*\* 2))

        j = np.argmax(sm)

        xr.extend([int(bit) for bit in bin(j)[2:].zfill(bps)])

    xr = np.array(xr)

    # Reshape and count errors

    x\_reshaped = x.reshape(-1)

    xr\_reshaped = xr[:len(x\_reshaped)]

    errors = np.sum(x\_reshaped != xr\_reshaped)

    return errors

def simulate\_ber(bps, Nsymb, ns, EbNo\_values, coherent=True):

    """

    Simulate Bit Error Rate (BER) over a range of Eb/No values.

    Parameters:

    - bps: Bits per symbol

    - Nsymb: Number of symbols

    - ns: Samples per symbol

    - EbNo\_values: List of Eb/No values in dB

    - coherent: Boolean indicating coherent or non-coherent detection

    Returns:

    - ber: List of BER values corresponding to Eb/No values

    """

    ber = []

    for EbNo in EbNo\_values:

        if coherent:

            errors = fsk\_errors(bps, Nsymb, ns, EbNo)

        else:

            errors = fsk\_errors\_non\_coh(bps, Nsymb, ns, EbNo)

        ber.append(errors / (Nsymb \* bps))

    return ber

# Define the BER values directly inside the functions

def theoretical\_ber\_coh(EbNo, M):

    """

    Return theoretical BER values for coherent FSK.

    Parameters:

    - EbNo: Array of Eb/No values in linear scale

    - M: Modulation order

    Returns:

    - BER: Array of BER values

    """

    if M == 2:

        return np.array([0.15866, 0.13093, 0.10403, 0.078896, 0.056495,

    0.037679, 0.023007, 0.012587, 0.0060044, 0.0024133,

    0.0007827, 0.00019399, 3.4303e-05, 3.9692e-06, 2.6951e-07])

    elif M == 4:

        return np.array([0.11814, 0.087789, 0.060786, 0.038512, 0.021824,

    0.010751, 0.0044428, 0.0014733, 0.00037102, 6.6229e-05,

    7.6892e-06, 5.2118e-07, 1.7997e-08, 2.6653e-10, 1.362e-12])

    elif M == 8:

        return np.array([0.10227, 0.067834, 0.041318, 0.022024, 0.0099156,

    0.0036087, 0.0010058, 0.00020086, 2.6486e-05, 2.0836e-06,

    8.6119e-08, 1.5924e-09, 1.074e-11, 2.0391e-14, 7.8524e-18])

    elif M == 16:

        return np.array([0.089859, 0.055663, 0.029957, 0.013469, 0.004819,

    0.001293, 0.00024205, 2.897e-05, 1.9921e-06, 6.8928e-08,

    1.0145e-09, 5.1257e-12, 6.7629e-15, 1.6453e-18, 4.6157e-23])

    elif M == 32:

        return np.array([0.082719, 0.047105, 0.022469, 0.0085348, 0.0024266,

    0.00047917, 6.0083e-05, 4.2975e-06, 1.5388e-07, 2.3425e-09,

    1.2294e-11, 1.6992e-14, 4.3826e-18, 1.3266e-22, 2.1688e-28])

    else:

        raise ValueError("Unsupported value of M for coherent case")

def theoretical\_ber\_non\_coh(EbNo, M):

    """

    Return theoretical BER values for non-coherent FSK.

    Parameters:

    - EbNo: Array of Eb/No values in linear scale

    - M: Modulation order

    Returns:

    - BER: Array of BER values

    """

    if M == 2:

        return np.array([0.30327, 0.26644, 0.22637, 0.18438, 0.1424, 0.10287,

    0.068311, 0.0408, 0.021324, 0.0094212, 0.003369,

    0.0009231, 0.00018089, 2.3244e-05, 1.7558e-06])

    elif M == 4:

        return np.array([0.22934, 0.18475, 0.13987, 0.097719, 0.061557,

    0.033946, 0.01579, 0.0059139, 0.0016837, 0.00033939,

    4.4371e-05, 3.3753e-06, 1.3045e-07, 2.1593e-09, 1.233e-11])

    elif M == 8:

        return np.array([0.19472, 0.14559, 0.099187, 0.059806, 0.030757,

    0.012878, 0.0041438, 0.00095467, 0.00014449, 1.2945e-05,

    6.0428e-07, 1.2541e-08, 9.4638e-11, 2.0092e-13, 8.6607e-17])

    elif M == 16:

        return np.array([0.17469, 0.12169, 0.074737, 0.038861, 0.01625,

    0.005127, 0.0011288, 0.00015786, 1.2538e-05, 4.943e-07,

    8.2001e-09, 4.6432e-11, 6.8517e-14, 1.8682e-17, 6.0826e-22])

    elif M == 32:

        return np.array([0.16103, 0.10471, 0.058014, 0.025984, 0.0088058,

    0.0020817, 0.00031127, 2.6219e-05, 1.0859e-06, 1.8789e-08,

    1.1086e-10, 1.7154e-13, 4.9582e-17, 1.737e-21, 4.2722e-27])

    else:

        raise ValueError("Unsupported value of M for non-coherent case")

def update\_plot(bps):

    """

    Update the BER plot based on the selected bits per symbol.

    Parameters:

    - bps: Bits per symbol

    """

    # Start timer and display loading animation

    loader\_html2.value = loading

    start\_time = time.time()

    Nsymb = 2000             # Number of symbols

    ns = 100                 # Samples per symbol

    EbNo\_dB\_sim = np.arange(0, 10, 2)    # Eb/No values for simulation

    EbNo\_dB\_theory = np.arange(0, 15, 1) # Eb/No values for theoretical curves

    EbNo\_sim = 10 \*\* (EbNo\_dB\_sim / 10)        # Convert dB to linear scale

    EbNo\_theory = 10 \*\* (EbNo\_dB\_theory / 10)

    M = 2 \*\* bps                                # Modulation order

    # Simulate BER for coherent and non-coherent detection

    ber\_coh\_sim = simulate\_ber(bps, Nsymb, ns, EbNo\_dB\_sim, coherent=True)

    ber\_non\_coh\_sim = simulate\_ber(bps, Nsymb, ns, EbNo\_dB\_sim, coherent=False)

    # Get theoretical BER values

    ber\_coh\_theory = theoretical\_ber\_coh(EbNo\_theory, M)

    ber\_non\_coh\_theory = theoretical\_ber\_non\_coh(EbNo\_theory, M)

    # Plot BER curves

    plt.figure(figsize=(10, 6))

    plt.semilogy(EbNo\_dB\_sim, ber\_coh\_sim, 'o', label=f'Simulated Coherent {M}-FSK')

    plt.semilogy(EbNo\_dB\_sim, ber\_non\_coh\_sim, 's', label=f'Simulated Non-Coherent {M}-FSK')

    plt.semilogy(EbNo\_dB\_theory, ber\_coh\_theory, label=f'Theoretical Coherent {M}-FSK')

    plt.semilogy(EbNo\_dB\_theory, ber\_non\_coh\_theory, label=f'Theoretical Non-Coherent {M}-FSK')

    plt.title(f'{M}-FSK System')

    plt.xlabel('Eb/No (dB)')

    plt.ylabel('Bit Error Rate (BER)')

    plt.grid(True, which='both')

    plt.legend()

    plt.show()

    # Display elapsed time and completion checkmark

    elapsed\_time = time.time() - start\_time

    timer\_html2.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html2.value = done

# Dropdown widget for selecting bits per symbol

bps\_dropdown = Dropdown(

    options=[('2-FSK', 1), ('4-FSK', 2), ('8-FSK', 3), ('16-FSK', 4), ('32-FSK', 5)],

    value=2,

    description='M-FSK:',

    style={'description\_width': 'initial'},

    continuous\_update=False

)

# Group the loader and timer widgets

loader\_timer\_box = widgets.VBox(

    [loader\_html2, timer\_html2],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Input widgets container

input\_widgets = widgets.VBox([bps\_dropdown])

# Combine input widgets and loader/timer into an HBox

ui = widgets.HBox(

    [input\_widgets, loader\_timer\_box],

    layout=widgets.Layout(align\_items='center')

)

# Create interactive output

out = widgets.interactive\_output(update\_plot, {'bps': bps\_dropdown})

# Display UI and output

clear\_output(wait=True)

display(ui, out)

Σχεδίαση Φάσματος Ζωνοπερατού Σήματος για Σύστημα 32-FSK

# Loading animation and completion checkmark HTML code

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey border \*/

                                     border-top: 12px solid #01cc97; /\* Blue border on top \*/

                                     border-radius: 50%;             /\* Circular shape \*/

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite; /\* Spinning animation \*/'>

        </div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }     /\* Start rotation \*/

        100% { transform: rotate(360deg); } /\* End rotation \*/

    }

    </style>

    """

done = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>  <!-- Checkmark symbol -->

    </div>

    """

loader\_html3 = widgets.HTML(value=loading)

timer\_html3 = widgets.HTML(value="Elapsed time: - seconds")

def update\_plot(fsklvl, Nsymb, EbNo):

    """

    Plot the power spectral density for coherent and non-coherent FSK.

    Parameters:

    - fsklvl: FSK levels (e.g., 2, 4, 8, etc.)

    - Nsymb: Number of symbols

    - EbNo: Energy per bit to noise power spectral density ratio in dB

    """

    # Start timer and display loading animation

    loader\_html3.value = loading

    start\_time = time.time()

    clear\_output(wait=True)

    bps = int(math.log2(fsklvl))  # Bits per symbol

    M = 2 \*\* bps                  # Modulation order

    BR = 1                        # Baud Rate

    T = 1 / BR                    # Symbol duration

    fc = 2 \* M \* BR               # Carrier frequency

    ns = 80                       # Samples per symbol

    nb = bps \* Nsymb              # Total number of bits

    f = fc + (BR / 2) \* (np.arange(1, M + 1) - (M + 1) / 2)  # Frequencies for coherent FSK

    # Adjust 'ns' based on maximum frequency

    fmax = np.max(f)

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    Ts = T / ns

    # Calculate SNR in dB

    SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)

    # Generate random input bits

    y = np.random.randint(0, 2, nb)

    x = y.reshape((len(y) // bps, bps))

    # Time vectors

    t = np.arange(0, len(x) \* T, T)

    tks = np.arange(0, T, Ts)

    # Generate coherent FSK signal

    s = []

    A = np.sqrt(2 / T / ns)

    for k in range(len(x)):

        symbol\_bits = ''.join(map(str, x[k]))

        symbol\_index = int(symbol\_bits, 2)

        fk = f[symbol\_index]

        tk = (k \* T) + tks

        s.append(np.sin(2 \* np.pi \* fk \* tk))

    s\_coherent = np.concatenate(s)

    # Compute power spectral density

    frequencies\_coherent, Pxx\_coherent = welch(s\_coherent, fs=ns \* BR, nperseg=50000, noverlap=25000)

    # Parameters for non-coherent FSK

    ns = 90

    Ts = T / ns

    f = fc + BR \* (np.arange(1, M + 1) - (M + 1) / 2)  # Frequencies for non-coherent FSK

    fmax = np.max(f)

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    Ts = T / ns

    # Generate non-coherent FSK signal

    s = []

    for k in range(len(x)):

        symbol\_bits = ''.join(map(str, x[k]))

        symbol\_index = int(symbol\_bits, 2)

        fk = f[symbol\_index]

        tk = (k \* T) + tks

        s.append(np.sin(2 \* np.pi \* fk \* tk))

    s\_non\_coherent = np.concatenate(s)

    # Compute power spectral density

    frequencies\_non\_coherent, Pxx\_non\_coherent = welch(s\_non\_coherent, fs=ns \* BR, nperseg=50000)

    # Plot PSDs

    fig, axs = plt.subplots(2, 1, figsize=(10, 8))

    axs[0].semilogy(frequencies\_coherent, Pxx\_coherent, linewidth=0.5)

    axs[0].set\_title("Coherent FSK")

    axs[0].set\_xlabel("Frequency (Hz)")

    axs[0].set\_ylabel("Power Spectral Density (dB/Hz)")

    axs[0].grid(True)

    axs[1].semilogy(frequencies\_non\_coherent, Pxx\_non\_coherent, linewidth=0.5)

    axs[1].set\_title("Non-Coherent FSK")

    axs[1].set\_xlabel("Frequency (Hz)")

    axs[1].set\_ylabel("Power Spectral Density (dB/Hz)")

    axs[1].grid(True)

    plt.tight\_layout()

    plt.show()

    # Display elapsed time and completion checkmark

    elapsed\_time = time.time() - start\_time

    timer\_html3.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html3.value = done

# Widgets for user input

fsklvl\_dropdown = Dropdown(

    options=[2, 4, 8, 16, 32],

    value=4,

    description='FSK Levels:',

    style={'description\_width': 'initial'},

    layout=Layout(flex='1 1 auto', width='auto'),

    continuous\_update=False

)

Nsymb\_slider = IntSlider(

    value=10000,

    min=1000,

    max=50000,

    step=1000,

    description='Number of Symbols:',

    style={'description\_width': 'initial'},

    layout=Layout(flex='1 1 auto', width='auto'),

    continuous\_update=False

)

EbNo\_slider = IntSlider(

    value=8,

    min=0,

    max=20,

    step=1,

    description='Eb/No (dB):',

    style={'description\_width': 'initial'},

    layout=Layout(flex='1 1 auto', width='auto'),

    continuous\_update=False

)

# Group the loader and timer widgets

loader\_timer\_box = widgets.VBox(

    [loader\_html3, timer\_html3],

    layout=widgets.Layout(margin='0 0 0 20px', width='auto')

)

# Input widgets container

input\_widgets = widgets.VBox(

    [fsklvl\_dropdown, Nsymb\_slider, EbNo\_slider],

    layout=widgets.Layout(flex='1 1 auto', width='auto')

)

# Combine input widgets and loader/timer into an HBox

ui = widgets.HBox(

    [input\_widgets, loader\_timer\_box],

    layout=widgets.Layout(align\_items='center')

)

# Create interactive output

out = widgets.interactive\_output(update\_plot, {'fsklvl': fsklvl\_dropdown, 'Nsymb': Nsymb\_slider, 'EbNo': EbNo\_slider})

# Display UI and output

clear\_output(wait=True)

display(ui, out)

Εξομοίωση Συστήματος MSK σε Ζωνοπερατό Δίαυλο με 2 Mbps και Υπολογισμός BER για Eb/No=10 dB

# Loading animation and completion checkmark HTML code

loading = """

    <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

        <div class='loader' style='border: 12px solid #f3f3f3; /\* Light grey \*/

                                     border-top: 12px solid #01cc97; /\* Blue \*/

                                     border-radius: 50%;

                                     width: 40px;

                                     height: 40px;

                                     animation: spin 2s linear infinite;'></div>

    </div>

    <style>

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }

    }

    </style>

    """

done = """

        <div style='display: flex; justify-content: center; align-items: center; height: 80px;'>

            <div style='font-size: 40px; color: #01cc97;'>&#10003;</div>

        </div>

        """

loader\_html4 = widgets.HTML(

  value=loading

)

timer\_html4 = widgets.HTML(

    value="Elapsed time: - seconds"

)

# Function that simulates Additive White Gaussian Noise (AWGN)

def awgn(signal, SNR):

    """

    Additive White Gaussian Noise (AWGN) channel simulation.

    Parameters:

    - signal: Input signal

    - SNR: Signal-to-Noise Ratio in dB

    Returns:

    - noisy\_signal: Signal with added AWGN noise

    """

    snr\_linear = 10 \*\* (SNR / 10)

    signal\_power = np.mean(signal \*\* 2)

    noise\_power = signal\_power / snr\_linear

    noise = np.sqrt(noise\_power) \* np.random.normal(size=signal.shape)

    return signal + noise

# MSK error function

def msk\_errors(Nbits, nsamp, EbNo):

    """

    Simulate MSK modulation and calculate BER without precoding.

    Parameters:

    - Nbits: Number of bits

    - nsamp: Oversampling factor

    - EbNo: Energy per bit to noise power spectral density ratio in dB

    Returns:

    - BER: Bit Error Rate

    """

    n = Nbits             # Number of data bits

    R = 2000000           # Bit rate (2 Mbps)

    fc = 8000000          # Carrier frequency (8 MHz)

    ns = nsamp            # Oversampling factor

    T = 1 / R             # Bit duration

    Ts = T / ns           # Sampling interval

    fss = 1 / Ts          # Sampling frequency

    # Calculate SNR in dB

    SNR = EbNo - 10 \* np.log10(ns/2)  # in dB

    # Generate random input bits (-1 or 1)

    y = np.concatenate(([1], np.sign(np.random.rand(n - 1) - 0.5)))  # random numbers, -1 or 1

    x = y

    # Generate NRZ polar pulse train samples

    g = np.ones(ns)

    xx = upfirdn(g, x, up=ns)  # NRZ polar pulse train samples

    # Time vector

    ts = np.arange(0, len(xx) \* Ts, Ts)  # of length ns\*(n+1)

    # MSK Transmitter

    xs = xx

    theta = np.cumsum(xs) \* np.pi / 2 / ns

    xs\_i = np.cos(theta)    # in-phase component

    xs\_q = np.sin(theta)    # quadrature component

    # Ensure that xs\_i and xs\_q are the same length as the time grid `ts`

    if len(xs\_i) > len(ts):

        xs\_i = xs\_i[:len(ts)]

        xs\_q = xs\_q[:len(ts)]

    elif len(ts) > len(xs\_i):

        ts = ts[:len(xs\_i)]

    # Modulation

    s = xs\_i \* np.cos(2 \* np.pi \* fc \* ts) - xs\_q \* np.sin(2 \* np.pi \* fc \* ts)

    # Addition of noise

    s = awgn(s, SNR)

    ## MSK RECEIVER

    xs\_i = s \* np.cos(2 \* np.pi \* fc \* ts)

    xs\_q = -s \* np.sin(2 \* np.pi \* fc \* ts)

    # LP (Parks-McClellan) filter

    f1 = 0.75\*(fss/2)/ns

    f2 = 4\*f1

    order = 8 \* ns

    fpts = [0, f1, f2, fss/2]

    mag = [1, 1, 0, 0]

    wt = [1, 1]

    b = firwin2(order+1, fpts, mag, fs=fss)

    a = 1

    len\_xs\_i = len(xs\_i)

    dummy = np.concatenate((xs\_i, np.zeros(order)))

    dummy1 = lfilter(b, a, dummy)

    delay = order // 2

    xs\_i = dummy1[delay:delay + len\_xs\_i]

    xs\_i = np.concatenate((xs\_i, np.ones(nsamp-1)))

    dummy = np.concatenate((xs\_q, np.zeros(order)))

    dummy1 = lfilter(b, a, dummy)

    xs\_q = dummy1[delay:delay + len\_xs\_i]

    bi = 1

    xr\_1 = 1

    xr = np.zeros(n)

    for k in range(0, n, 2):

        li = np.arange((k+1) \* ns, min((k + 3) \* ns-1, len(xs\_i)))

        lq = np.arange(k \* ns, min((k + 2) \* ns-1, len(xs\_q)))

        xi = xs\_i[li]

        xq = xs\_q[lq]

        gmi = np.cos(np.pi / 2 / T \* Ts \* li)  # matched-filter pulse

        gmq = -gmi  # =sin(pi/2/T\*Ts\*lq);

        bi\_1 = bi

        bi = np.sign(np.sum(xi \* gmi))

        bq = np.sign(np.sum(xq \* gmq))

        xr[k] = bi\_1 \* bq

        xr[k+1] = bi \* bq

        xr\_1 = xr[k + 1]

    xr = xr.reshape(-1)

    err = np.not\_equal(x, xr)

    errors = np.sum(err)

    return errors / Nbits

# MSK error function with precoding

def msk\_errors\_precoding(Nbits, nsamp, EbNo):

    """

    Simulate MSK modulation and calculate BER with precoding.

    Parameters:

    - Nbits: Number of bits

    - nsamp: Oversampling factor

    - EbNo: Energy per bit to noise power spectral density ratio in dB

    Returns:

    - BER: Bit Error Rate

    """

    n = Nbits

    R = 2000000

    fc = 8000000

    ns = nsamp

    T = 1 / R

    Ts = T / ns

    fss = 1 / Ts

    # Calculate SNR

    SNR = EbNo - 10 \* np.log10(ns/2)  # in dB

    # Generate random input bits and apply precoding

    y = np.concatenate(([1], np.sign(np.random.rand(n - 1) - 0.5)))  # random numbers, -1 or 1

    x = y

    x[0] = 1

    for i in range(1, len(y)):

        x[i] = y[i] \* x[i-1]  # Apply precoding rule

    # Generate NRZ polar pulse train samples

    g = np.ones(ns)

    xx = upfirdn(g, x, up=ns)  # NRZ polar pulse train samples

    # Time grid

    ts = np.arange(0, len(xx) \* Ts, Ts)  # of length ns\*(n+1)

    ## MSK TRANSMITTER

    xs = xx

    theta = np.cumsum(xs) \* np.pi / 2 / ns

    xs\_i = np.cos(theta)  # in-phase component

    xs\_q = np.sin(theta)  # quadrature component

    # Ensure that xs\_i and xs\_q are the same length as the time grid ts

    if len(xs\_i) > len(ts):

        xs\_i = xs\_i[:len(ts)]

        xs\_q = xs\_q[:len(ts)]

    elif len(ts) > len(xs\_i):

        ts = ts[:len(xs\_i)]

    # Modulation

    s = xs\_i \* np.cos(2 \* np.pi \* fc \* ts) - xs\_q \* np.sin(2 \* np.pi \* fc \* ts)

    # Addition of noise

    s = awgn(s, SNR)

    ## MSK RECEIVER

    xs\_i = s \* np.cos(2 \* np.pi \* fc \* ts)

    xs\_q = -s \* np.sin(2 \* np.pi \* fc \* ts)

    # LP (Parks-McClellan) filter

    f1 = 0.75\*(fss/2)/ns

    f2 = 4\*f1

    order = 8 \* ns

    fpts = [0, f1, f2, fss/2]

    mag = [1, 1, 0, 0]

    wt = [1, 1]

    b = firwin2(order+1, fpts, mag, fs=fss)

    a = 1

    len\_xs\_i = len(xs\_i)

    dummy = np.concatenate((xs\_i, np.zeros(order)))

    dummy1 = lfilter(b, a, dummy)

    delay = order // 2

    xs\_i = dummy1[delay:delay + len\_xs\_i]

    xs\_i = np.concatenate((xs\_i, np.ones(nsamp-1)))

    dummy = np.concatenate((xs\_q, np.zeros(order)))

    dummy1 = lfilter(b, a, dummy)

    xs\_q = dummy1[delay:delay + len\_xs\_i]

    # Updated MSK decoding for precoded bits with recursive logic

    bi = 1

    xr\_1 = 1  # Initialize previous decoded bit (xr\_1)

    xr = np.zeros(n)  # Array to store decoded bits

    for k in range(0, n, 2):

        li = np.arange((k+1) \* ns, min((k + 3) \* ns-1, len(xs\_i)))

        lq = np.arange(k \* ns, min((k + 2) \* ns-1, len(xs\_q)))

        xi = xs\_i[li]

        xq = xs\_q[lq]

        # Matched filter output (to match MSK modulation characteristics)

        gmi = np.cos(np.pi / 2 / T \* Ts \* li)  # In-phase matched filter pulse

        gmq = -gmi  # Quadrature matched-filter pulse (sin is negative of cosine)

        # Save previous in-phase matched filter output

        bi\_1 = bi

        # Decode in-phase (I) and quadrature (Q) components

        bi = np.sign(np.sum(xi \* gmi))

        bq = np.sign(np.sum(xq \* gmq))

        # Apply recursive decoding rule for precoded MSK

        xr[k] = bi\_1 \* bq  # Decode the k-th bit

        xr[k+1] = bi \* bq  # Decode the (k+1)-th bit

        # Update the previously decoded bit (xr\_1)

        xr\_1 = xr[k+1]

    xr = xr.reshape(-1)

    err = np.not\_equal(y, xr)

    errors = np.sum(err)

    return 0.5\*errors / Nbits

# Function to simulate BER and plot

def update\_msk\_plot(nsamp):

    """

    Update the BER plot for MSK modulation with and without precoding.

    Parameters:

    - nsamp: Oversampling factor

    """

    # Start timer and display loading animation

    if loader\_html4.value != loading:

        loader\_html4.value = loading  # Set loading indicator only if not already set

    start\_time = time.time()

    EbNo\_range = np.arange(0, 9, 1)  # EbNo from 0 to 10 dB

    Nbits = 10000  # Increase number of bits to reduce variance

    simulated\_BER = []

    simulated\_BER\_precoding = []

    theoretical\_BER = []

    theoretical\_BER\_precoded = []

    for EbNo in EbNo\_range:

        # Simulate both with and without precoding

        sim\_BER = msk\_errors(Nbits, nsamp, EbNo)

        sim\_BER\_precoded = msk\_errors\_precoding(Nbits, nsamp, EbNo)

        # Theoretical BER without precoding

        theoretical\_BER\_value = 0.9 \* erfc(np.sqrt(10\*\*(EbNo / 10)))

        # Theoretical BER with precoding

        theoretical\_BER\_precoded\_value = erfc(np.sqrt(10\*\*(EbNo / 10))) / 2

        simulated\_BER.append(sim\_BER)

        simulated\_BER\_precoding.append(sim\_BER\_precoded)

        theoretical\_BER.append(theoretical\_BER\_value)

        theoretical\_BER\_precoded.append(theoretical\_BER\_precoded\_value)

    # Plot results

    plt.figure(figsize=(10, 6))

    plt.semilogy(EbNo\_range, simulated\_BER, 'o', label='Simulated BER (without precoding)')

    plt.semilogy(EbNo\_range, simulated\_BER\_precoding, 's', label='Simulated BER (with precoding)')

    plt.semilogy(EbNo\_range, theoretical\_BER, label='Theoretical BER (without precoding)')

    plt.semilogy(EbNo\_range, theoretical\_BER\_precoded, label='Theoretical BER (with precoding)')

    plt.xlabel('$E\_b/N\_0$ (dB)')

    plt.ylabel('Bit Error Rate (BER)')

    plt.title('MSK Modulation (with and without precoding)')

    plt.legend()

    plt.grid(True, which='both')

    plt.show()

    # Update timer and loading indicator

    elapsed\_time = time.time() - start\_time

    timer\_html4.value = f"Elapsed time: {elapsed\_time:.2f} seconds"

    loader\_html4.value = done

# Interactive UI components

nsamp\_slider = widgets.IntSlider(

    value=32,

    min=16,

    max=128,

    step=16,

    description='Oversampling Factor:',

    layout=Layout(width='auto', flex='1 1 auto'),

    style={'description\_width': 'initial'},

    continuous\_update=False

)

# Group the loader and timer together (they will appear next to each other horizontally)

loader\_timer\_box = widgets.VBox([loader\_html4, timer\_html4], layout=widgets.Layout(margin='0 0 0 20px', width='auto'))

# Create a VBox for the input widgets (similar to the first refactored code)

input\_widgets = widgets.VBox([nsamp\_slider], layout=widgets.Layout(flex='1 1 auto', width='auto'))

# Create an HBox to combine inputs and loader timer box, with the same layout style

ui = widgets.HBox([input\_widgets, loader\_timer\_box], layout=widgets.Layout(align\_items='center'))

# Create the interactive output for the update\_msk\_plot function

out = widgets.interactive\_output(update\_msk\_plot, {'nsamp': nsamp\_slider})

# Display the UI and output

clear\_output(wait=True)  # Clear the previous output

display(ui, out)