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

**Διαδραστική προσομοίωση Ψηφιακών Επικοινωνιών με ελεύθερο λογισμικό**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΡΑΒΑΝΤΙΝΟΥ-ΚΑΡΛΑΤΟΥ ΣΩΤΗΡΗ

Επιβλέπων

Μήτρου Νικόλαος

Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2024

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

**Διαδραστική προσομοίωση Ψηφιακών Επικοινωνιών με ελεύθερο λογισμικό**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΡΑΒΑΝΤΙΝΟΥ-ΚΑΡΛΑΤΟΥ ΣΩΤΗΡΗ

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, Οκτώβριος 2024

.............................   
Αραβαντινός-Καρλάτος Γ. Σωτήρης   
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Αραβαντινός-Καρλάτος Σωτήρης, 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 εστιάζει στην ανάπτυξη μιας ιστοσελίδας με διαδραστικές δυνατότητες, η οποία παρέχει προσομοιώσεις συστημάτων ψηφιακής επικοινωνίας μέσω χρήσης ελεύθερου λογισμικού. Το έργο βασίζεται στη θεωρητική βάση των συστημάτων ψηφιακής επικοινωνίας, από τις σημειώσεις του μαθήματος Ψηφιακές Επικοινωνίες Ι, και προχωρά σε αναπαραστάσεις και προσομοιώσεις των βασικών κεφαλαίων που αφορούν την ψηφιακή επεξεργασία σήματος, τις τεχνικές ανίχνευσης και διαμόρφωσης, όπως και τα χαρακτηριστικά του φάσματος των ψηφιακών κυματομορφών.

Η εργασία περιλαμβάνει ανασκόπηση των εργαλείων λογισμικού που χρησιμοποιήθηκαν, αναλύει τα κριτήρια επιλογής τους και τεκμηριώνει τα πλεονεκτήματά τους σε προσομοιώσεις. Επιπλέον, καταγράφονται οι μεθοδολογίες και οι τεχνικές υλοποίησης που εφαρμόστηκαν για τη διαδραστική απεικόνιση της θεωρίας. Στα αποτελέσματα, παρουσιάζονται οι προσομοιώσεις με συγκρίσεις των αποτελεσμάτων με τις θεωρητικές αναλύσεις, καθώς και οι προκλήσεις που προέκυψαν.

Τέλος, η εργασία προτείνει μελλοντικές επεκτάσεις, όπως η εισαγωγή νέων εργαλείων και εκπαιδευτικών δραστηριοτήτων, ενώ παρέχει αναλυτική τεκμηρίωση του κώδικα και οδηγούς χρήσης στο παράρτημα.

**Λέξεις-Κλειδιά:** Ψηφιακές Επικοινωνίες, Προσομοίωση, Διαδραστική Ιστοσελίδα, Ελεύθερο Λογισμικό, Ψηφιακή Επεξεργασία Σήματος, Ψηφιακή Διαμόρφωση, QAM, PSK, FSK, MSK, Matched Filters, Χαρακτηριστικά Φάσματος, Jupyter Book, Python

.

# Abstract

This thesis focuses on the development of a website with interactive capabilities that provides simulations of digital communication systems using open-source software. The project covers the theoretical foundation of digital communication systems, based on the notes from the course Digital Communications I, and progresses to representations and simulations of the key chapters related to digital signal processing, detection, modulation techniques, and the spectral characteristics of digital waveforms.

The thesis includes a review of the software tools used, analyzes the criteria for their selection, and documents their advantages and performance in simulations. Furthermore, the methodologies and implementation techniques used for the interactive representation of the theory are outlined, with emphasis on the simulations developed for each chapter. The results section presents the simulations with comparisons of the outcomes with theoretical analyses, as well as the challenges encountered.

Finally, the thesis suggests future extensions, such as the introduction of new tools and educational activities, while also providing detailed code documentation and user guides in the appendix..

**Keywords:** Digital Communications, Simulation, Interactive Website, Open-Source Software, Digital Signal Processing, Digital Modulation, QAM, PSK, FSK, MSK, Matched Filters, Spectral Characteristics, Jupyter Book, Python.

# Ευχαριστίες

Αρχικά, θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες προς τον επιβλέποντα καθηγητή μου, κ. Νικόλαο Μήτρου, για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου αυτή τη διπλωματική εργασία. Μέσα από αυτό το έργο, μου δόθηκε η ευκαιρία να εμβαθύνω σε ένα άκρως ενδιαφέρον και πολύτιμο αντικείμενο, που αφορά την ανάπτυξη μιας ιστοσελίδας με διαδραστικές δυνατότητες συστημάτων ψηφιακής επικοινωνίας χρησιμοποιώντας ελεύθερο λογισμικό. Θερμές ευχαριστίες αξίζουν και στην κ. Κωνσταντία Σακκά για την αδιάλειπτη υποστήριξη και καθοδήγησή της καθ' όλη τη διάρκεια της εκπόνησης αυτής της εργασίας.

Αυτή η διπλωματική εργασία σηματοδοτεί την ολοκλήρωση των προπτυχιακών μου σπουδών στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Καθώς φτάνω στο τέλος αυτής της πορείας, νιώθω την ανάγκη να εκφράσω τις θερμές μου ευχαριστίες στους γονείς μου, Γεράσιμο και Μαρία, για την αμέριστη στήριξή τους καθ’ όλη τη διάρκεια των σπουδών μου. Η υποστήριξή τους ήταν καθοριστική για την πρόοδό μου. Επίσης, ευχαριστώ τα αδέλφια μου που ήταν πάντα δίπλα μου σε αυτό το ταξίδι.

Τέλος, θα ήθελα να εκφράσω την ευγνωμοσύνη μου προς όλους τους φίλους που στάθηκαν στο πλευρό μου κατά τη διάρκεια των φοιτητικών μου χρόνων. Μαζί μοιραστήκαμε τις δυσκολίες των μαθημάτων, τις προκλήσεις των εργασιών και τις απαιτητικές εξεταστικές περιόδους, δημιουργώντας πολύτιμες αναμνήσεις που θα με συνοδεύουν για πάντα. Σας ευχαριστώ από καρδιάς όλους.

# Οργάνωση του τόμου

Ο παρών τόμος είναι οργανωμένος σε επτά κύρια κεφάλαια, τα οποία καλύπτουν την ανάπτυξη μιας διαδραστικής ιστοσελίδας με προσομοιώσεις συστημάτων ψηφιακής επικοινωνίας, καθώς και τη θεωρητική και πρακτική τους προσέγγιση μέσω ελεύθερου λογισμικού.

Στο πρώτο κεφάλαιο παρουσιάζεται η εισαγωγή στην εργασία, οι στόχοι που τέθηκαν, και η συνολική δομή της. Στο δεύτερο κεφάλαιο γίνεται μια ανασκόπηση της βιβλιογραφίας, με αναφορές στις σημειώσεις του μαθήματος Ψηφιακές Επικοινωνίες Ι και στις ενότητες που θα γίνουν διαδραστικές, καλύπτοντας θέματα όπως η ψηφιακή επεξεργασία σήματος, η ανίχνευση μέσω προσαρμοσμένων φίλτρων και οι διαμορφώσεις L-ASK, QAM, PSK, FSK και MSK.

Το τρίτο κεφάλαιο επικεντρώνεται στη μεθοδολογία που ακολουθήθηκε για την υλοποίηση των διαδραστικών προσομοιώσεων, με έμφαση στη μετάβαση από τη MATLAB στην Python και τη χρήση Jupyter Notebooks μέσα σε ένα Jupyter Book. Το τέταρτο κεφάλαιο παρουσιάζει τα αποτελέσματα των προσομοιώσεων και αναλύει τις προκλήσεις που προέκυψαν κατά τη διάρκεια της υλοποίησης.

Στο πέμπτο κεφάλαιο εξετάζονται οι μελλοντικές επεκτάσεις της εργασίας, με την εισαγωγή νέων σημειώσεων και εργαλείων που μπορούν να ενσωματωθούν. Το έκτο κεφάλαιο συνοψίζει το σύνολο της εργασίας παρουσιάζοντας περιληπτικά κάθε κομμάτι της, ενώ το έβδομο κεφάλαιο καταγράφει τις πηγές που χρησιμοποιήθηκαν για τη θεωρητική και πρακτική υποστήριξη της εργασίας.

Στα παραρτήματα παρέχονται αναλυτικοί οδηγοί εγκατάστασης των εργαλείων που χρησιμοποιήθηκαν, καθώς και τεκμηρίωση του κώδικα και οδηγίες χρήσης της τελικής διαδραστικής εφαρμογής που αναπτύχθηκε.

# Πίνακας περιεχομένων

[Περίληψη 2](#_Toc179297057)

[Abstract 3](#_Toc179297058)

[Ευχαριστίες 2](#_Toc179297059)

[Οργάνωση του τόμου 2](#_Toc179297060)

[Πίνακας περιεχομένων 3](#_Toc179297061)

[Κατάλογος Σχημάτων 11](#_Toc179297062)

[Εισαγωγή 12](#_Toc179297063)

[1. Στόχοι της Εργασίας 13](#_Toc179297064)

[2. Δομή της Εργασίας 14](#_Toc179297065)

[Ανασκόπηση της Βιβλιογραφίας 15](#_Toc179297066)

[1. Οι σημειώσεις του μαθήματος 16](#_Toc179297067)

[Chapter 1: Digital Signal Processing in Telecommunications 16](#_Toc179297068)

[Chapter 3: Optimal digital detection – Matched filters 18](#_Toc179297069)

[Chapter 4: Spectral Characteristics of Digital Waveforms & Nyquist signaling 19](#_Toc179297070)

[Chapter 5: Digital Modulation QAM and PSK 20](#_Toc179297071)

[Chapter 6: Digital Modulation FSK and MSK 21](#_Toc179297072)

[2. Εργαλεία που χρησιμοποιήθηκαν 22](#_Toc179297073)

[Εισαγωγή 22](#_Toc179297074)

[Διερεύνηση Εργαλείων 22](#_Toc179297075)

[Κριτήρια Επιλογής 28](#_Toc179297076)

[Επιλογή Εργαλείων 30](#_Toc179297077)

[Μεθοδολογία, Σχεδίαση και Υλοποίηση 41](#_Toc179297078)

[1. Εισαγωγή 42](#_Toc179297079)

[2. Μέθοδος υλοποίησης του συνολικού project 42](#_Toc179297080)

[3. Υλοποίηση διαδραστικών στοιχείων 44](#_Toc179297081)

[Chapter 1: Digital Signal Processing in Telecommunications 44](#_Toc179297082)

[Chapter 3: Optimal digital detection – Matched filters 49](#_Toc179297083)

[Chapter 4: Spectral Characteristics of Digital Waveforms & Nyquist signaling 50](#_Toc179297084)

[Chapter 5: Digital Modulation QAM and PSK 51](#_Toc179297085)

[Chapter 6: Digital Modulation FSK and MSK 52](#_Toc179297086)

[Αποτελέσματα 53](#_Toc179297087)

[1. Εισαγωγή 54](#_Toc179297088)

[2. Αποτελέσματα Προσομοιώσεων 55](#_Toc179297089)

[Chapter 1: Digital Signal Processing in Telecommunications 55](#_Toc179297090)

[Chapter 3: Optimal digital detection – Matched filters 62](#_Toc179297091)

[Chapter 4: Spectral Characteristics of Digital Waveforms & Nyquist signaling 64](#_Toc179297092)

[Chapter 5: Digital Modulation QAM and PSK 66](#_Toc179297093)

[Chapter 6: Digital Modulation FSK and MSK 69](#_Toc179297094)

[3. Συμπεράσματα 71](#_Toc179297095)

[Μελλοντικές επεκτάσεις 72](#_Toc179297096)

[1. Επέκταση σε Εργαστηριακές Ασκήσεις 73](#_Toc179297097)

[2. Εισαγωγή νέων εργαλείων 73](#_Toc179297098)

[Σύνοψη 74](#_Toc179297099)

[1. Συνοπτική παρουσίαση του σκοπού 75](#_Toc179297100)

[2. Περίληψη κύριων βημάτων και μεθοδολογίας 76](#_Toc179297101)

[3. Βασικά αποτελέσματα 77](#_Toc179297102)

[4. Συμπεράσματα 78](#_Toc179297103)

[Πηγές 79](#_Toc179297104)

[Παράρτημα Ι 80](#_Toc179297105)

[1. Πλήρης Οδηγός Εγκατάστασης 80](#_Toc179297106)

[Παράρτημα ΙΙ 84](#_Toc179297107)

[1. User Guide for Jupyter Book with Sphinx-Thebe 84](#_Toc179297108)

[Εκκίνηση Διαδραστικού Κώδικα 84](#_Toc179297109)

[Αλλαγή Εργαστηριακής Άσκησης ή Εργαλείου BER 85](#_Toc179297110)

[Διαδραστικότητα Στα Διαγράμματα 86](#_Toc179297111)

[Αν Δεν Φαίνεται το Αρχικό Διάγραμμα 86](#_Toc179297112)

[Αναμονή Χωρίς Αποτελέσματα 86](#_Toc179297113)

[Επανεκκίνηση και Εκτέλεση Όλων 86](#_Toc179297114)

[2. Documentation: Basic Elements of a Jupyter Book 87](#_Toc179297115)

[Δομή του Jupyter Book 88](#_Toc179297116)

[Αρχεία Περιεχομένου 89](#_Toc179297117)

[Ρυθμίσεις του Jupyter Book 89](#_Toc179297118)

[Προσθήκη Κώδικα 91](#_Toc179297119)

[Διαδραστικότητα με το Thebe 92](#_Toc179297120)

[Δημιουργία και Ανάπτυξη του Βιβλίου 92](#_Toc179297121)

[Συμπέρασμα 92](#_Toc179297122)

# Κατάλογος Σχημάτων

**No table of figures entries found.**

Κεφάλαιο 1

# Εισαγωγή

## **Στόχοι της Εργασίας**

Οι στόχοι της παρούσας εργασίας επικεντρώνονται στην αναζήτηση, αξιολόγηση, επιλογή και χρήση κατάλληλων εργαλείων που θα υποστηρίξουν τη δημιουργία μιας διαδραστικής ιστοσελίδας για την προσομοίωση συστημάτων ψηφιακών επικοινωνιών. Η πλατφόρμα αυτή στοχεύει να προσφέρει στους φοιτητές τη δυνατότητα να εξερευνούν και να κατανοούν καλύτερα τις αρχές των ψηφιακών επικοινωνιών, μέσα από πρακτικές εφαρμογές και πειραματισμό.

Ένας βασικός στόχος της εργασίας είναι να εντοπιστούν εργαλεία που προσφέρουν επαρκή διαδραστικότητα και ευκολία χρήσης, διατηρώντας παράλληλα την ευελιξία που απαιτείται για την προσομοίωση και ανάλυση συστημάτων σε πραγματικό χρόνο. Η έρευνα επικεντρώνεται σε εργαλεία ανοιχτού κώδικα, τα οποία επιτρέπουν την ενσωμάτωση διαδραστικών στοιχείων σε εκπαιδευτικά περιβάλλοντα.

Η γλώσσα προγραμματισμού Python, λόγω της ευρείας χρήσης της στις επιστήμες των δεδομένων και της ανάλυσης, αποτελεί κεντρικό εργαλείο στην υλοποίηση των στόχων αυτών. Τέλος, απαραίτητο στοιχείο θα αποτελέσει η ενσωμάτωση kernel στην ιστοσελίδα για τη ζωντανή εκτέλεση του κώδικα Python απευθείας στον φυλλομετρητή, εξασφαλίζοντας, έτσι, για τους χρήστες, έναν εύκολο και άμεσο τρόπο αλληλεπίδρασης με τις προσομοιώσεις σε πραγματικό χρόνο.

## Δομή της Εργασίας

Η δομή της παρούσας εργασίας βασίζεται σε μια οργανωμένη και συστηματική προσέγγιση της έρευνας, αξιολόγησης και επιλογής εργαλείων, που θα χρησιμοποιηθούν για την ανάπτυξη μιας διαδραστικής ιστοσελίδας για την προσομοίωση συστημάτων ψηφιακών επικοινωνιών. Η εργασία συνδέει θεωρητικά και πρακτικά εργαλεία που υποστηρίζουν την επίτευξη των στόχων της, όπως είναι η παροχή ενός εκπαιδευτικού περιβάλλοντος για την καλύτερη κατανόηση των βασικών αρχών των ψηφιακών επικοινωνιών μέσω πρακτικών εφαρμογών.

Στα αρχικά κεφάλαια, παρουσιάζεται η έρευνα και η αξιολόγηση των διαθέσιμων εργαλείων ανοιχτού κώδικα, που θα μπορούσαν να χρησιμοποιηθούν για την ανάπτυξη της πλατφόρμας. Ενώ έπειτα, γίνεται ανάλυση των εργαλείων που τελικά επιλέχθηκαν, με λεπτομέρειες για τις δυνατότητες που προσφέρουν στην υλοποίηση διαδραστικών προσομοιώσεων.

Παρακάτω, περιγράφεται η μεθοδολογία σχεδίασης και υλοποίησης της διαδραστικής πλατφόρμας, καταγράφοντας τους τρόπους με τους οποίους τα εργαλεία χρησιμοποιήθηκαν για την ανάπτυξη προσομοιώσεων για κάθε κεφάλαιο. Ύστερα, παρουσιάζονται τα αποτελέσματα από τις προσομοιώσεις και τις προκλήσεις που αντιμετωπίστηκαν.

Επιπλέον, αναφερόμαστε σε μελλοντικές επεκτάσεις της εργασίας, οι οποίες περιλαμβάνουν την προσθήκη των εργαστηριακών ασκήσεων από το μάθημα Ψηφιακές Επικοινωνίες Ι, καθώς και την ενσωμάτωση νέων εργαλείων που θα βελτιώσουν τη λειτουργικότητα της πλατφόρμας και τη μαθησιακή εμπειρία.

Στο τέλος της εργασίας, παρατίθενται οι αναφορές, ένας οδηγός εγκατάστασης εργαλείων, καθώς και η τεκμηρίωση του κώδικα και οι οδηγίες χρήσης για τις προσομοιώσεις που αναπτύχθηκαν.

Κατά αυτόν τον τρόπο, η εργασία οικοδομεί μια συνεκτική σύνδεση μεταξύ θεωρίας και πράξης, προκειμένου να επιτευχθεί η δημιουργία ενός πλήρως λειτουργικού διαδραστικού περιβάλλοντος μάθησης για τις ψηφιακές επικοινωνίες, συνδυάζοντας τις βέλτιστες πρακτικές από τα δύο πεδία.

Κεφάλαιο 2

# Ανασκόπηση της Βιβλιογραφίας

## Οι σημειώσεις του μαθήματος

Η ενότητα αυτή περιγράφει συνοπτικά τα βασικά κεφάλαια της θεωρίας του μαθήματος Ψηφιακές Επικοινωνίες Ι. Σε κάθε κεφάλαιο, προσδιορίζεται ο στόχος της θεωρητικής ανάλυσης και επισημαίνονται τα τμήματα θεωρίας για τα οποία, με βάση τις αντίστοιχες ερωτήσεις στις εργαστηριακές ασκήσεις, θα πρέπει να βρεθούν εργαλεία προσομοίωσής τους με διαδραστικό τρόπο.

Αυτά τα στοιχεία θα παρέχουν στους χρήστες τη δυνατότητα να αλληλεπιδρούν με τα συστήματα ψηφιακών επικοινωνιών σε πραγματικό χρόνο, αναλύοντας και τροποποιώντας τα σήματα, βοηθώντας έτσι στην κατανόηση της συμπεριφοράς τους σε διάφορες συνθήκες.

Καλύπτονται κρίσιμα θέματα, όπως η ψηφιακή επεξεργασία σήματος, οι βέλτιστες τεχνικές ανίχνευσης μέσω φίλτρων προσαρμογής, τα χαρακτηριστικά φάσματος ψηφιακών κυματομορφών, καθώς και οι διάφορες τεχνικές διαμόρφωσης ASK, QAM, PSK, FSK και MSK.

### Chapter 1: Digital Signal Processing in Telecommunications

#### Σύνοψη κεφαλαίου

Το κεφάλαιο αυτό εισάγει τη διαδικασία ψηφιοποίησης και την ψηφιακή επεξεργασία σημάτων στο πεδίο των τηλεπικοινωνιών. Αναλύονται τα βασικά ζητήματα που σχετίζονται με τη μετάβαση από το συνεχές στο διακριτό πεδίο μέσω της δειγματοληψίας, όπως η δημιουργία φασματικών εικόνων και το πρόβλημα του aliasing. Παρουσιάζεται ο Διακριτός Μετασχηματισμός Fourier, ως η φασματική αναπαράσταση σημάτων διακριτού χρόνου στο διακριτό πεδίο συχνοτήτων, και ο αναγνώστης εισάγεται στον σχεδιασμό και την υλοποίηση ψηφιακών φίλτρων. Δίνονται παραδείγματα και εργαστηριακές ασκήσεις για την ανάλυση φάσματος και τη χρήση ψηφιακών φίλτρων. Στα παραρτήματα περιλαμβάνονται βασικά θεωρήματα και ιδιότητες των μετασχηματισμών Fourier στον συνεχή και διακριτό χρόνο, σε συνοπτική μορφή πίνακα.

#### Κομμάτια που θα κάνουμε διαδραστικά

* Δημιουργία & Οπτικοποίηση σημάτων: Παραγωγή σημάτων για την προσομοίωση συστημάτων ψηφιακών επικοινωνιών και δημιουργία γραφημάτων για να απεικονιστούν τα χρονικά και φασματικά διαγράμματα των σημάτων.
* Διαδραστική επεξεργασία μεταβλητών σήματος: Με τη χρήση διαδραστικών εργαλίων, να μπορούν οι χρήστες να τροποποιούν τις παραμέτρους του σήματος και να παρατηρούν σε πραγματικό χρόνο τις αλλαγές.
* Προσθήκη θορύβου: Δυνατότητα προσθήκης θορύβου στο σήμα για παρακολούθηση των επιδράσεών του στην απόδοση της επικοινωνίας.
* Ανάλυση Φάσματος: Δημιουργία γραφημάτων φασματικής ανάλυσης.
* Απεικόνιση Φίλτρου (Filter Visualization): Δυνατότητα επιλογής διαφορετικών φίλτρων και οπτική αναπαράσταση της απόκρισης τους με μορφή stem plot.
* Ανάλυση Συχνοτήτων Φίλτρων (Filter Frequency Response): Διαδραστική απεικόνιση της φασματικής απόκρισης φίλτρων για διαφορετικά μήκη φίλτρων και παραθύρων.
* Ανάλυση Equiripple Φίλτρων: Δυνατότητα επιλογής equiripple φίλτρων διαφορετικών μηκών και σύγκριση της φασματικής τους απόκρισης.
* Εφαρμογή Χαμηλοπερατού Φίλτρου (Low Pass Filter Application): Εφαρμογή φίλτρων χαμηλής διέλευσης και απεικόνιση της απόκρισης συχνοτήτων του φιλτραρισμένου σήματος.
* Φασματική Πυκνότητα Ισχύος (Power Spectral Density): Διαδραστική σύγκριση φίλτρων equiripple και ανάλυση της φασματικής πυκνότητας ισχύος των φιλτραρισμένων σημάτων.
* Ανάλυση Απόκρισης Bandpass Φίλτρων: Εφαρμογή φίλτρων bandpass και οπτική απεικόνιση της απόκρισης τους τόσο στην συχνότητα όσο και στην φασματική πυκνότητα του φιλτραρισμένου σήματος.
* Εφαρμογή Parks-McClellan Φίλτρου: Σχεδιασμός φίλτρου bandpass με τη μέθοδο Parks-McClellan για επίτευξη συγκεκριμένης εξασθένισης στα stop bands.
* Εφαρμογή Φίλτρου με 2 passbands: Σχεδιασμός και υλοποίηση φίλτρου με δύο passbands στο MATLAB, με απεικόνιση της φασματικής του απόκρισης.

### Chapter 3: Optimal digital detection – Matched filters

#### Σύνοψη κεφαλαίου

Αυτό το κεφάλαιο παρουσιάζει τη μέθοδο βέλτιστης αναγνώρισης παλμών, οι οποίοι μεταφέρουν ψηφιακά δεδομένα και έχουν υποστεί παραμόρφωση κατά τη μετάδοση λόγω προσθετικού λευκού θορύβου. Η μέθοδος υλοποιείται μέσω του λεγόμενου προσαρμοσμένου φίλτρου (matched filter), το οποίο λειτουργεί ουσιαστικά ως συσχετιστής σήματος. Στον δέκτη, μια σειρά προσαρμοσμένων φίλτρων (που αντιστοιχούν στους διάφορους, γραμμικά ανεξάρτητους παλμούς του συμφωνημένου ρεπερτορίου μετάδοσης - γνωστό ως αστερισμός σημάτων) υπολογίζει τη συσχέτιση κάθε παλμού με τον εισερχόμενο παλμό (μέσω πολλαπλασιασμού και ολοκλήρωσης σε κάθε βασική περίοδο, Τ). Το φίλτρο με την υψηλότερη έξοδο υποδεικνύει τον παλμό με την μεγαλύτερη πιθανότητα εκπομπής (δηλαδή, την μεγαλύτερη πιθανότητα να έχει σταλεί, με βάση το ληφθέν σήμα).

Μετά την παρουσίαση των βασικών χαρακτηριστικών των προσαρμοσμένων φίλτρων, γίνεται γενίκευση σε Ν-διάστατους γραμμικούς χώρους σημάτων. Στη συνέχεια, εξετάζεται συνοπτικά η ανάλυση σφαλμάτων και δίνονται παραδείγματα αναγνώρισης ημιτονικών παλμών (FSK) και διαμόρφωσης πλάτους παλμού (PAM ή ASK).

#### Κομμάτια που θα κάνουμε διαδραστικά

* Δημιουργία τυχαίων σημάτων: Δημιουργία τυχαίων σημάτων και παρουσίαση του ιστογράμματός τους.
* Εφαρμογή θορύβου: Επίδραση του θορύβου στο σήμα με τη ρύθμιση του λόγου Eb/No μέσω ενός διαδραστικού slider και τον υπολογισμό του ιστόγραμματος για διάφορες τιμές.
* ASK Bit Error Rate (BER) Visualization: Η διαδραστική δυνατότητα σύγκρισης διαμορφώσεων Μ-ASK με επιλογές για πειραματική και θεωρητική τιμή του BER. Να μπορούν οι χρήστες να επιλέξουν μεταξύ διαφορετικών διαμορφώσεων, δειγμάτων ανά σύμβολο, και τύπου matched filter για να δουν τα αποτελέσματα σε πραγματικό χρόνο.
* Ανάλυση με φίλτρα: Οπτικοποιήσεις απόκρισης φίλτρων πάνω σε ορθογώνιο και ημιτονοειδές σήμα.

### Chapter 4: Spectral Characteristics of Digital Waveforms & Nyquist signaling

#### Σύνοψη κεφαλαίου

Στόχος αυτού του κεφαλαίου είναι η ανάλυση των χαρακτηριστικών φάσματος των ψηφιακών κυματομορφών και η παρουσίαση της σηματοδότησης Nyquist, η οποία χρησιμοποιείται για την ελαχιστοποίηση των παραμορφώσεων λόγω της διασποράς και την αποφυγή του φαινομένου της διασυμβολικής παρεμβολής. Το κεφάλαιο εισάγει τις αρχές της σηματοδότησης Nyquist και εξετάζει τα σχετικά θέματα, όπως το εύρος ζώνης και την αποτελεσματικότητα των σημάτων. Περιλαμβάνονται διαδραστικές ασκήσεις που εξετάζουν την αναπαράσταση του φάσματος των ψηφιακών κυματομορφών και την ανάλυση των συστημάτων που βασίζονται στη σηματοδότηση Nyquist, εστιάζοντας στη διασφάλιση της μέγιστης απόδοσης των σημάτων χωρίς παραμόρφωση.

#### Κομμάτια που θα κάνουμε διαδραστικά

* Ρυθμιζόμενες παράμετροι: Εισαγωγή διαδραστικού τρόπου ρύθμισης παραμέτρων μήκους bitstream, roll-off factor, nsamp (δείγματα ανά σύμβολο), group delay, και σειρά του φίλτρου.
* Οπτικοποίηση σημάτων και φίλτρων: Δημιουργία γραφημάτων για την παρουσίαση του φίλτρου, του φιλτραρισμένου σήματος, τη σύγκριση μεταξύ των αρχικών και φιλτραρισμένων σημάτων και την ανάλυση της φασματικής πυκνότητας ισχύος του ληφθέντος σήματος.
* BER καμπύλες: Με δυνατότητα επιλογής επιπέδου M-ASK, εμφάνισης των θεωρητικών και πειραματικών καμπυλών σε σχέση με το Eb/N0 και σύγκριση μεταξύ κωδικοποίησης Gray και Natural.

### Chapter 5: Digital Modulation QAM and PSK

#### Σύνοψη κεφαλαίου

Το κεφάλαιο αυτό εξετάζει δύο από τις πιο συχνά χρησιμοποιούμενες μεθόδους ψηφιακής διαμόρφωσης, τη διαμόρφωση QAM (Τετραγωνική Διαμόρφωση Πλάτους) και τη διαμόρφωση PSK (Διαμόρφωση Μετατόπισης Φάσης). Παρουσιάζει τη δομή του πομπού και του δέκτη για κάθε διαμόρφωση, την ανάλυση σφαλμάτων σε περιβάλλον λευκού θορύβου και τα φασματικά χαρακτηριστικά των σημάτων. Ο στόχος είναι η κατανόηση του τρόπου με τον οποίο τα συστήματα αυτά αξιοποιούνται σε συνθήκες πραγματικού θορύβου και η επίτευξη υψηλότερων ταχυτήτων μετάδοσης. Οι φοιτητές θα συμμετέχουν σε προσομοιώσεις που περιλαμβάνουν την υλοποίηση και ανάλυση σημάτων QAM και PSK, καθώς και την κατανόηση των επιπτώσεων του θορύβου και του φάσματος σε αυτά τα συστήματα διαμόρφωσης.

#### Κομμάτια που θα κάνουμε διαδραστικά

* QAM και PSK Constellations: Διαδραστικά διαγράμματα αστερισμών για L-QAM και M-PSK. Οι χρήστες να μπορούν να επιλέξουν το επίπεδο διαμόρφωσης και να δουν τις αντίστοιχες θέσεις των συμβόλων στον αστερισμό.
* QAM και PSK Καμπύλες BER: Δημιουργία γραφημάτων για τις καμπύλες BER τόσο για QAM όσο και για PSK διαμορφώσεις, όπου οι χρήστες μπορούν να συγκρίνουν την θεωρητική και πειραματική καμπύλη και να ρυθμίσουν παραμέτρους όπως το roll-off, τις συχνότητες f1 και f2, και το bitrate για την ανάλυση της πιθανότητας σφάλματος.
* Διαδραστική Φασματική Πυκνότητα Ισχύος: Δυνατότητα οπτικοποίησης της φασματικής πυκνότητας ισχύος για διάφορες διαμορφώσεις QAM και συχνοτήτων f1 και f2.
* Διαδραστικός Υπολογισμός Παραμέτρων: Δυνατότητα υπολογισμού παραμέτρων, με βάση τις παραμέτρους (R, M και roll-off) που επιλέγει ο χρήστης, όπως ο ρυθμός.

### Chapter 6: Digital Modulation FSK and MSK

#### Σύνοψη κεφαλαίου

Ο σκοπός του Το κεφάλαιο αυτό επικεντρώνεται στις τεχνικές ψηφιακής διαμόρφωσης FSK (Διαμόρφωση Μετατόπισης Συχνότητας) και MSK (Ελάχιστη Μετατόπιση Φάσης). Εξετάζονται οι βασικές αρχές αυτών των διαμορφώσεων, η συμπεριφορά τους σε συνθήκες θορύβου, οι πιθανότητες σφαλμάτων και τα φασματικά τους χαρακτηριστικά. Η MSK παρουσιάζεται ως μια ειδική περίπτωση της FSK που επιτυγχάνει ελάχιστο εύρος ζώνης και μεγαλύτερη αποδοτικότητα στην επικοινωνία. Οι φοιτητές θα εργαστούν σε προσομοιώσεις που αφορούν τη διαμόρφωση FSK και MSK, θα παρατηρήσουν τις φασματικές ιδιότητες των σημάτων και θα αξιολογήσουν την απόδοση αυτών των τεχνικών διαμόρφωσης υπό την επίδραση θορύβου.

#### Κομμάτια που θα κάνουμε διαδραστικά

* Αριθμός σφαλμάτων για Eb/No: Γράφημα στο οποίο ο χρήστης ρυθμίζει ένα εύρος του Eb/No, και στη συνέχεια εμφανίζεται η καμπύλη που απεικονίζει τον αριθμό των σφαλμάτων σε σχέση με το Eb/No για συστήματα FSK.
* FSK Bit Error Rate: Διαδραστική σύγκριση των καμπυλών BER για coherent και non coherent συστήματα FSK. Οι χρήστες μπορούν να δουν τις καμπύλες που αντιπροσωπεύουν την θεωρητική και πειραματική απόδοση.
* Ανάλυση Φάσματος coherent και non coherent FSK: Διαδραστικό γράφημα ανάλυσης της φασματικής πυκνότητας ισχύος τόσο για coherent όσο και για non coherent FSK συστήματα, όπου οι χρήστες μπορούν να προσαρμόσουν τις παραμέτρους όπως ο αριθμός συμβόλων και το Eb/No.
* MSK Bit Error Rate: Διαδραστικό γράφημα για σύγκριση μεταξύ των καμπυλών BER με και χωρίς precoding για τη διαμόρφωση MSK. Οι χρήστες μπορούν να δουν τις θεωρητικές και πειραματικές καμπύλες, συγκρίνοντας τις επιδόσεις για διαφορετικές τιμές Eb/No.

## Εργαλεία

### Εισαγωγή

|  |  |
| --- | --- |
| GitHub |  |

Για την οργάνωση του γενικότερου πλαισίου του έργου, την αποθήκευση των αρχείων και την τελική ανάπτυξη της διαδικτυακής εφαρμογής, χρησιμοποιήθηκε το GitHub (GitHub - Documentation, n.d.). Η πλατφόρμα αυτή επέτρεψε τη διαχείριση του κώδικα και την ασφαλή αποθήκευση των αρχείων του έργου, ενώ μέσω της υπηρεσίας GitHub Pages πραγματοποιήθηκε το τελικό deployment της εφαρμογής, προσφέροντας έναν εύκολο και αξιόπιστο τρόπο για τη δημοσίευση του περιεχομένου στο διαδίκτυο.

### Διερεύνηση Εργαλείων

Για την επίτευξη των παραπάνω διαδραστικών δυνατοτήτων θα χρειαστεί να βρεθούν εργαλεία γύρω από τις εξής κατηγορίες:

* Διαδραστική πλατφόρμα παρουσίασης του τελικού αποτελέσματος.
* Ενσωμάτωση kernel για εκτέλεση κώδικα σε πραγματικό χρόνο.
* Βιβλιοθήκες python για την προσομοίωση συστημάτων ψηφιακής επικοινωνίας.
* Γραφική αναπαράσταση δεδομένων.
* Διαδραστικοί μέθοδοι εισαγωγής παραμέτρων.

#### Διαδραστική πλατφόρμα παρουσίασης του τελικού αποτελέσματος.

Για την παρουσίαση του τελικού αποτελέσματος με διαδραστικό τρόπο, υπάρχουν διάφορες πλατφόρμες και εργαλεία που επιτρέπουν την ενσωμάτωση κώδικα, γραφημάτων και διαδραστικών στοιχείων σε ένα ενιαίο και προσβάσιμο περιβάλλον. Ακολουθεί η λίστα από αυτά που δοκιμάσαμε:

|  |  |
| --- | --- |
| 1. Jupyter Book: Εργαλείο για τη δημιουργία βιβλίων και τεκμηρίωσης από Jupyter Notebooks και αρχεία Markdown, με υποστήριξη διαδραστικών στοιχείων.   (Jupyter Book - Documentation, n.d.) |  |
| 1. Voila: Μετατρέπει Jupyter Notebooks σε αυτόνομες διαδραστικές web εφαρμογές χωρίς την ανάγκη γραφής JavaScript.   (Voila - Documentation, n.d.) |  |
| 1. MkDocs: Στατικός site generator που μετατρέπει αρχεία Markdown σε όμορφα τεκμηριωμένα ιστοσελίδες.   (MkDocs - Documentation, n.d.) |  |
| 1. Dash: Πλαίσιο για τη δημιουργία διαδραστικών web εφαρμογών για ανάλυση δεδομένων με Python.   (Dash - Documentation, n.d.) |  |
| 1. Google Colab: Διαδικτυακό περιβάλλον για την εκτέλεση Jupyter Notebooks, με υποστήριξη για Python και ενσωμάτωση με το Google Drive.   (Google Colab - Documentation, n.d.) |  |

#### Ενσωμάτωση kernel για εκτέλεση κώδικα σε πραγματικό χρόνο.

Για την ενσωμάτωση ενός kernel, που επιτρέπει την εκτέλεση κώδικα σε πραγματικό χρόνο μέσα από ιστοσελίδες, προχωρήσαμε μετά την επιλογή του Jupyter Book ως πλατφόρμα δομής της εφαρμογής μας. Έτσι, δοκιμάσαμε τις παρακάτω επιλογές που μας προσέφερε:

Διερεύνηση Εργαλείων

Μετά την επιλογή του Jupyter Book ως πλατφόρμας για τη δομή της εφαρμογής μας, εξετάσαμε τις επιλογές που προσφέρει το Jupyter Book για την ενσωμάτωση ενός kernel που επιτρέπει την εκτέλεση κώδικα σε πραγματικό χρόνο μέσα από ιστοσελίδες. Οι επιλογές που παρέχει το Jupyter Book είναι οι εξής:

|  |  |
| --- | --- |
| 1. notebook\_interface: Επιτρέπει την ενσωμάτωση των Jupyter Notebooks στο βιβλίο, παρέχοντας τη δυνατότητα προβολής του κώδικα και των αποτελεσμάτων του. Ωστόσο, δεν υποστηρίζει την εκτέλεση κώδικα σε πραγματικό χρόνο από τον χρήστη μέσα από την ιστοσελίδα. (Jupyter Book - Notebook Interface, n.d.) |  |
| 1. binderhub\_url: Παρέχει τη δυνατότητα σύνδεσης με ένα BinderHub server, επιτρέποντας την εκτέλεση των notebooks σε ένα απομακρυσμένο περιβάλλον. Οι χρήστες μπορούν να ανοίξουν τα notebooks σε ένα περιβάλλον Binder και να εκτελέσουν τον κώδικα. (Jupyter Book - Binder, n.d.) |  |
| 1. jupyterhub\_url: Επιτρέπει τη σύνδεση με ένα JupyterHub server, όπου οι χρήστες μπορούν να εκτελούν τα notebooks σε ένα κοινόχρηστο περιβάλλον εργασίας. (Jupyter Book - JupyterHub, n.d.) |  |
| 1. thebe: Μια JavaScript βιβλιοθήκη που μετατρέπει στατικές σελίδες HTML σε διαδραστικές, επιτρέποντας την εκτέλεση κώδικα απευθείας μέσα από την ιστοσελίδα. Το Thebe συνδέεται με έναν Jupyter kernel μέσω ενός backend, όπως το Binder. (Jupyter Book - Thebe, n.d.) |  |
| 1. colab\_url: Παρέχει συνδέσμους που επιτρέπουν στους χρήστες να ανοίξουν τα notebooks στο Google Colab, όπου μπορούν να εκτελέσουν τον κώδικα σε ένα cloud-based περιβάλλον. (Jupyter Book – Google Colab, n.d.) |  |

#### Βιβλιοθήκες python συστημάτων ψηφιακής επικοινωνίας

Για την προσομοίωση ψηφιακών συστημάτων στην Python, υπάρχουν αρκετές βιβλιοθήκες που προσφέρουν πλούσιες λειτουργίες και εργαλεία. Ακολουθεί η λίστα από αυτές που δοκιμάσαμε:

|  |  |
| --- | --- |
| 1. NumPy: Παρέχει υποστήριξη για πολυδιάστατους πίνακες και μαθηματικές συναρτήσεις υψηλής απόδοσης. (NumPy - Documentation, n.d.) |  |
| 1. SciPy: Προσφέρει ένα ευρύ φάσμα επιστημονικών και τεχνικών υπολογισμών, συμπεριλαμβανομένων σημάτων και επεξεργασίας εικόνων. (SciPy - Documentation, n.d.) | C:\Users\Sotiris\AppData\Local\Microsoft\Windows\INetCache\Content.Word\scipy.png |
| 1. scipy.signal: Υπο-πακέτο του SciPy, εξειδικευμένο στην επεξεργασία σημάτων, φίλτρων και συστημάτων. (scipy.signal - Documentation, n.d.) | C:\Users\Sotiris\AppData\Local\Microsoft\Windows\INetCache\Content.Word\scipy.png |
| 1. CommPy: Βιβλιοθήκη για την προσομοίωση τηλεπικοινωνιακών συστημάτων, συμπεριλαμβανομένων διαμορφώσεων, καναλιών και κωδικοποίησης. (CommPy - Documentation, n.d.) |  |
| 1. math: Ενσωματωμένη βιβλιοθήκη της Python με βασικές μαθηματικές συναρτήσεις. (Math Module - Documentation, n.d.) |  |
| 1. Matplotlib: Βιβλιοθήκη για γραφική απεικόνιση δεδομένων σε 2D και 3D. (Matplotlib - Documentation, n.d.) |  |
| 1. SimPy: Βιβλιοθήκη για την προσομοίωση διακριτών γεγονότων, χρήσιμη για συστήματα επικοινωνίας. (SimPy - Documentation, n.d.) |  |
| 1. PyWavelets: Βιβλιοθήκη για ανάλυση wavelet, χρήσιμη στην επεξεργασία σημάτων. (PyWavelets - Documentation, n.d.) |  |
| 1. scikit-dsp-comm: Παρέχει εργαλεία για επεξεργασία ψηφιακών σημάτων και επικοινωνιών. (scikit-dsp-comm - Documentation, n.d.) |  |
| 1. PySDR: Συνοδευτική βιβλιοθήκη για το βιβλίο "Python SDR", με εργαλεία για Software Defined Radio. (PySDR GitHub, n.d.) |  |

#### Γραφική αναπαράσταση δεδομένων

Από την πληθώρα βιβλιοθηκών Python για τη γραφική αναπαράσταση δεδομένων, έγινε έρευνα στις παρακάτω:

|  |  |
| --- | --- |
| 1. Matplotlib: Μια από τις πιο δημοφιλείς βιβλιοθήκες για τη δημιουργία στατικών, κινούμενων και διαδραστικών γραφημάτων στην Python. (Matplotlib - Documentation, n.d.) |  |
| 1. Seaborn: Βιβλιοθήκη που βασίζεται στο Matplotlib και προσφέρει υψηλού επιπέδου διεπαφές για στατιστικά γραφήματα. (Seaborn - Documentation, n.d.) |  |
| 1. Plotly: Βιβλιοθήκη για διαδραστικά γραφήματα που μπορούν να ενσωματωθούν σε ιστοσελίδες. (Plotly - Documentation, n.d.) |  |
| 1. Bokeh: Παρέχει δυνατότητες για διαδραστικά γραφήματα σε ιστοσελίδες μέσω Python. (Bokeh - Documentation, n.d.) |  |
| 1. PyPlot: Μέρος του Matplotlib. Εύκολη στη χρήση διεπαφή για τη δημιουργία γραφημάτων, παρόμοια με το MATLAB. (PyPlot - Documentation, n.d.) | **C:\Users\Sotiris\AppData\Local\Microsoft\Windows\INetCache\Content.Word\matplotlib.png** |
| 1. ggplot: Βιβλιοθήκη εμπνευσμένη από το ggplot2 της R, για δηλωτική δημιουργία γραφημάτων. (ggplot - Documentation, n.d.) |  |
| 1. Altair: Βιβλιοθήκη για δηλωτική στατιστική οπτικοποίηση βασισμένη στο Vega-Lite. (Altair - Documentation, n.d.) |  |
| 1. Pandas Visualization: Παρέχει απλές μεθόδους για τη δημιουργία γραφημάτων απευθείας από δεδομένα Pandas. (Pandas Visualization - Documentation, n.d.) |  |
| 1. Holoviews: Διευκολύνει τη δημιουργία σύνθετων γραφημάτων με λιγότερο κώδικα. (Holoviews - Documentation, n.d.) |  |
| 1. VisPy: Βιβλιοθήκη για γρήγορη οπτικοποίηση δεδομένων χρησιμοποιώντας GPU. (VisPy - Documentation, n.d.) |  |

#### Διαδραστικοί μέθοδοι εισαγωγής παραμέτρων.

Για την υλοποίηση διαδραστικών μεθόδων εισαγωγής παραμέτρων στην Python, σε περιβάλλοντα όπως τα Jupyter Notebooks, εξετάσαμε τα ακόλουθα εργαλεία:

|  |  |
| --- | --- |
| 1. IPython: Προηγμένη διεπαφή για την Python που παρέχει διαδραστικό περιβάλλον εργασίας, συμπεριλαμβανομένων δυνατοτήτων όπως τα Jupyter Notebooks. (IPython - Documentation, n.d.) |  |
| 1. ipywidgets: Βιβλιοθήκη που επιτρέπει τη δημιουργία διαδραστικών widget σε Jupyter Notebooks, διευκολύνοντας την εισαγωγή παραμέτρων και την αλληλεπίδραση με τον κώδικα. (ipywidgets - Documentation, n.d.) |  |
| 1. Bokeh: Βιβλιοθήκη για τη δημιουργία διαδραστικών οπτικοποιήσεων σε ιστοσελίδες, υποστηρίζοντας widgets για εισαγωγή παραμέτρων. (Bokeh - Documentation, n.d.) |  |
| 1. Plotly Dash: Πλατφόρμα για τη δημιουργία διαδραστικών web εφαρμογών με Python, επιτρέποντας την εισαγωγή παραμέτρων μέσω διαφόρων στοιχείων διεπαφής. (Plotly Dash - Documentation, n.d.) |  |
| 1. Streamlit: Πλαίσιο για τη δημιουργία web εφαρμογών για μηχανική μάθηση και επιστήμη δεδομένων με απλή σύνταξη Python. (Streamlit - Documentation, n.d.) |  |
| 1. Voila: Επιτρέπει τη μετατροπή Jupyter Notebooks σε διαδραστικές web εφαρμογές χωρίς την ανάγκη γραφής κώδικα JavaScript. (Voila - Documentation, n.d.) |  |
| 1. Panel: Βιβλιοθήκη για τη δημιουργία διαδραστικών dashboards και εφαρμογών, υποστηρίζοντας πολλαπλές βιβλιοθήκες γραφικών. (Panel - Documentation, n.d.) |  |
| 1. Gradio: Επιτρέπει τη δημιουργία απλών διεπαφών χρήστη για μηχανική μάθηση και επιστήμη δεδομένων. (Gradio - Documentation, n.d.) |  |
| 1. PySimpleGUI: Παρέχει μια απλή διεπαφή για τη δημιουργία GUI εφαρμογών με Python. (PySimpleGUI - Documentation, n.d.) |  |
| 1. Tkinter: Ενσωματωμένη βιβλιοθήκη της Python για τη δημιουργία GUI εφαρμογών. (Tkinter - Documentation, n.d.) |  |

### Κριτήρια Επιλογής

Για την επιλογή των κατάλληλων εργαλείων, λάβαμε υπόψη τα ακόλουθα κριτήρια:

* Συμβατότητα με τις Απαιτήσεις της Προσομοίωσης: Πρέπει να παρέχει τις απαραίτητες λειτουργίες για την προσομοίωση των ψηφιακών συστημάτων που καλύπτονται στο μάθημα.
* Ευκολία Χρήσης: Το εργαλείο πρέπει να είναι εύκολο στην εκμάθηση και χρήση, με απλή σύνταξη και κατανοητές λειτουργίες.
* Δυνατότητες Διαδραστικότητας: Η δυνατότητα δημιουργίας διαδραστικών εφαρμογών που επιτρέπουν στον χρήστη να αλληλεπιδρά με τα δεδομένα σε πραγματικό χρόνο.
* Υποστήριξη Κοινότητας: Μια ενεργή κοινότητα χρηστών και προγραμματιστών εξασφαλίζει την συνεχή βελτίωση του εργαλείου και την επίλυση προβλημάτων.
* Διαθεσιμότητα Τεκμηρίωσης: Η ύπαρξη εκτενούς και κατανοητής τεκμηρίωσης είναι απαραίτητη για την αποτελεσματική χρήση του εργαλείου.
* Απόδοση και Ταχύτητα: Οι βιβλιοθήκες πρέπει να είναι αποδοτικές και να επιτρέπουν την επεξεργασία μεγάλων όγκων δεδομένων σε εύλογο χρόνο.
* Συμβατότητα: Η δυνατότητα ενσωμάτωσης με άλλα εργαλεία για την ανάπτυξη της ολοκληρωμένης εφαρμογής.
* Υποστήριξη Markdown και Κώδικα: Η δυνατότητα συγγραφής σε Markdown και η ενσωμάτωση κώδικα είναι απαραίτητη για την παρουσίαση της θεωρίας και των παραδειγμάτων.
* Δυνατότητες Οργάνωσης Περιεχομένου: Πρέπει να επιτρέπει την οργάνωση του υλικού σε κεφάλαια, ενότητες και υποενότητες, με εύκολη πλοήγηση.
* Δυνατότητες Δημοσίευσης: Η πλατφόρμα πρέπει να επιτρέπει την εύκολη δημοσίευση του υλικού στο διαδίκτυο, ώστε να είναι προσβάσιμο από τους φοιτητές.
* Ωριμότητα και Σταθερότητα: Προτίμηση εργαλείων που έχουν δοκιμαστεί στον χρόνο και θεωρούνται σταθερά.

### Επιλογή Εργαλείων

Με βάση τα παραπάνω κριτήρια, και έπειτα από εκτενής δοκιμές πάνω σε κάθε εργαλείο, επιλέξαμε να χρησιμοποιήσουμε τα ακόλουθα:

#### Διαδραστική πλατφόρμα παρουσίασης του τελικού αποτελέσματος.

|  |  |
| --- | --- |
| Jupyter Book  (Jupyter Book - Documentation, n.d.) |  |

Γιατί το επιλέξαμε:

* Ενσωμάτωση με Jupyter Notebooks: Το Jupyter Book σχεδιάστηκε για να μετατρέπει Jupyter Notebooks, αρχεία Markdown και αρχεία MyST Markdown (μία επέκταση του προηγούμενου) σε ένα συνεκτικό βιβλίο, επιτρέποντας την άμεση αξιοποίηση του ήδη υπάρχοντος υλικού.
* Υποστήριξη Διαδραστικότητας: Επιτρέπει την ενσωμάτωση διαδραστικών στοιχείων, όπως τα ipywidgets, και υποστηρίζει την εκτέλεση κώδικα μέσα στις σελίδες του βιβλίου.
* Εύκολη Οργάνωση Περιεχομένου: Παρέχει εργαλεία για την οργάνωση του υλικού σε κεφάλαια και ενότητες, με αυτόματη δημιουργία πίνακα περιεχομένων και πλοήγησης.
* Δυνατότητες Προσαρμογής: Προσφέρει θέματα και επιλογές διαμόρφωσης για την προσαρμογή της εμφάνισης και της λειτουργικότητας του βιβλίου.
* Εύκολη Δημοσίευση: Μπορεί να δημοσιευτεί σε διάφορες πλατφόρμες, όπως το GitHub Pages, καθιστώντας το προσβάσιμο μέσω web browser.
* Υποστήριξη Κοινότητας: Ως μέρος του οικοσυστήματος Jupyter, έχει μια ενεργή κοινότητα και συνεχή ανάπτυξη.

Πλεονεκτήματα:

* Συνεκτική Παρουσίαση: Ενοποιεί κείμενο, κώδικα και διαδραστικά στοιχεία σε ένα ενιαίο περιβάλλον.
* Υποστήριξη Εκτελέσιμου Κώδικα: Επιτρέπει στους χρήστες να εκτελούν κώδικα μέσα στο βιβλίο, ενισχύοντας την διαδραστικότητα.
* Προσαρμογή Δομής: Επιτρέπει την προσαρμογή των συνδέσμων και της δομής του βιβλίου για καλύτερη πλοήγηση.
* Δυνατότητα Χρήσης Επεκτάσεων: Μπορεί να επεκταθεί με πρόσθετες λειτουργίες μέσω επεκτάσεων (π.χ. sphinx-inline tabs, thebe)

**Πώς συνέβαλε στην επίτευξη του στόχου της εργασίας**

* Ενοποίηση Υλικού: Το Jupyter Book μας επέτρεψε να συγκεντρώσουμε όλα τα κεφάλαια, τις σημειώσεις, τον κώδικα και τα διαδραστικά στοιχεία σε ένα ενιαίο και εύκολα προσβάσιμο βιβλίο.
* Διαδραστική Μάθηση: Μέσω του live code run που θα εξηγηθεί μετά. Οι φοιτητές μπορούν να αλληλεπιδρούν με τα διαδραστικά στοιχεία, να τροποποιούν παραμέτρους και να βλέπουν τα αποτελέσματα σε πραγματικό χρόνο μέσα από το βιβλίο.
* Ευκολία Συντήρησης και Ενημέρωσης: Οι ενημερώσεις στο περιεχόμενο και τον κώδικα μπορούν να γίνονται εύκολα και να δημοσιεύονται άμεσα.
* Υποστήριξη Συνεργασίας: Η χρήση του Git για τη διαχείριση του περιεχομένου επιτρέπει τη συνεργασία μεταξύ των εκπαιδευτών και τη συνεισφορά από πολλαπλούς χρήστες.
* Δυνατότητα Επεκτασιμότητας: Το Jupyter Book μπορεί να επεκταθεί και να προσαρμοστεί για μελλοντικές ανάγκες, όπως η προσθήκη νέων κεφαλαίων ή διαδραστικών στοιχείων.

**Συμπέρασμα**

Η επιλογή του Jupyter Book ως πλατφόρμας παρουσίασης του τελικού αποτελέσματος ανταποκρίνεται πλήρως στις ανάγκες της εργασίας. Μας επέτρεψε να δημιουργήσουμε ένα ολοκληρωμένο, διαδραστικό και εύκολα προσβάσιμο εκπαιδευτικό υλικό, που ενισχύει την κατανόηση των ψηφιακών επικοινωνιών και παρέχει στους φοιτητές ένα πολύτιμο εργαλείο μάθησης.

#### Ενσωμάτωση kernel για εκτέλεση κώδικα σε πραγματικό χρόνο.

Αφού αξιολογήσαμε τις διαθέσιμες επιλογές, επιλέξαμε να χρησιμοποιήσουμε το Thebe, δημιουργώντας έναν kernel στο Binder.

|  |  |
| --- | --- |
| Thebe  (Jupyter Book - Thebe, n.d.) |  |

Γιατί το επιλέξαμε:

* Διαδραστικότητα μέσα από το Jupyter Book: Το Thebe ενσωματώνεται εύκολα στο Jupyter Book, επιτρέποντας τη μετατροπή των στατικών κελιών κώδικα σε διαδραστικά. Οι χρήστες μπορούν να εκτελούν και να τροποποιούν τον κώδικα απευθείας μέσα από την ιστοσελίδα.
* Ευκολία Ενσωμάτωσης: Απαιτεί ελάχιστες αλλαγές στο αρχείο διαμόρφωσης του Jupyter Book. Με την προσθήκη `use\_thebe: true`, ενεργοποιείται η λειτουργικότητα του Thebe.
* Υποστήριξη Python και Βιβλιοθηκών: Λειτουργεί με Jupyter kernels, υποστηρίζοντας πλήρως την Python και τις απαιτούμενες βιβλιοθήκες μας, όπως NumPy, SciPy και Matplotlib.
* Εμπειρία Χρήστη: Οι χρήστες δεν χρειάζεται να εγκαταστήσουν πρόσθετο λογισμικό ή να φύγουν από την ιστοσελίδα. Η αλληλεπίδραση με τον κώδικα γίνεται ομαλά και άμεσα.

|  |  |
| --- | --- |
| Binder  (Jupyter Book - Binder, n.d.) |  |

Γιατί το επιλέξαμε:

* Σύνδεση με Thebe: Το Binder μπορεί να χρησιμοποιηθεί ως backend για το Thebe, παρέχοντας τον απαραίτητο Jupyter kernel για την εκτέλεση του κώδικα.
* Ευκολία Ρύθμισης: Η δημιουργία ενός περιβάλλοντος στο Binder γίνεται μέσω ενός public Git repository, όπου καθορίζουμε τις εξαρτήσεις και τις βιβλιοθήκες που χρειάζονται.
* Δωρεάν Υπηρεσία: Το mybinder.org προσφέρει δωρεάν hosting, καθιστώντας το ιδανικό για εκπαιδευτικούς σκοπούς και επιτρέποντας την ευρεία πρόσβαση χωρίς κόστος.
* Αναπαραγωγιμότητα: Διασφαλίζει ότι όλοι οι χρήστες εκτελούν τον κώδικα σε ένα σταθερό και ελεγχόμενο περιβάλλον, αποφεύγοντας προβλήματα συμβατότητας.

**Πώς συνέβαλαν στην επίτευξη του στόχου της εργασίας**

* Διαδραστική Εκτέλεση Κώδικα: Με το Thebe, οι φοιτητές μπορούν να εκτελούν κώδικα σε πραγματικό χρόνο μέσα από τις σελίδες του Jupyter Book, τροποποιώντας παραμέτρους και παρατηρώντας άμεσα τις επιπτώσεις στα αποτελέσματα και τα γραφήματα.
* Ενσωμάτωση με το Jupyter Book: Η λύση λειτουργεί άψογα με τη δομή και το περιεχόμενο που έχουμε δημιουργήσει, χωρίς την ανάγκη σημαντικών αλλαγών ή πρόσθετων ρυθμίσεων.
* Ευκολία Πρόσβασης: Οι φοιτητές χρειάζονται μόνο έναν web browser για να έχουν πλήρη πρόσβαση στο υλικό και τη διαδραστική λειτουργικότητα, διευκολύνοντας την εξ αποστάσεως μάθηση.
* Αναπαραγωγιμότητα και Συνέπεια: Το Binder εξασφαλίζει ότι όλοι οι χρήστες εκτελούν τον κώδικα στο ίδιο περιβάλλον, με τις ίδιες βιβλιοθήκες και εκδόσεις, παρέχοντας συνεπή αποτελέσματα.
* Ασφάλεια: Το Binder παρέχει ένα ασφαλές περιβάλλον εκτέλεσης, απομονώνοντας τον κώδικα των χρηστών και προστατεύοντας τους διακομιστές από κακόβουλες ενέργειες.
* Υποστήριξη Κοινότητας: Η ενεργή κοινότητα γύρω από το Thebe και το Binder μας επέτρεψε να αξιοποιήσουμε παραδείγματα, να επιλύσουμε προβλήματα και να προσαρμόσουμε τη λύση στις ανάγκες μας.

**Συμπέρασμα**

Με την επιλογή του Thebe και τη δημιουργία ενός kernel στο Binder, καταφέραμε να ενσωματώσουμε έναν διαδραστικό kernel στο Jupyter Book, επιτρέποντας την εκτέλεση κώδικα σε πραγματικό χρόνο μέσα από τις ιστοσελίδες μας. Αυτή η λύση ανταποκρίνεται πλήρως στις ανάγκες μας για ένα ευέλικτο, προσβάσιμο και διαδραστικό εκπαιδευτικό εργαλείο, ενισχύοντας την εμπειρία μάθησης των φοιτητών και διευκολύνοντας την κατανόηση των ψηφιακών συστημάτων επικοινωνιών.

#### Βιβλιοθήκες python συστημάτων ψηφιακής επικοινωνίας

|  |  |
| --- | --- |
| 1. Numpy   (NumPy - Documentation, n.d.) |  |

Γιατί το επιλέξαμε: Αποτελεί τη βάση για επιστημονικούς υπολογισμούς στην Python. Παρέχει αποδοτικούς πολυδιάστατους πίνακες και λειτουργίες που είναι απαραίτητες για την επεξεργασία σημάτων.

Πλεονεκτήματα:

* Υψηλή απόδοση σε αριθμητικούς υπολογισμούς.
* Ευρεία χρήση και μεγάλη κοινότητα.
* Εύκολη ενσωμάτωση με άλλες βιβλιοθήκες.

|  |  |
| --- | --- |
| 1. SciPy   (SciPy - Documentation, n.d.) |  |

Γιατί το επιλέξαμε: Προσφέρει προηγμένες επιστημονικές λειτουργίες που δεν υπάρχουν στο NumPy, ιδιαίτερα στον τομέα της επεξεργασίας σημάτων.

Πλεονεκτήματα:

* Πλούσια συλλογή αλγορίθμων και λειτουργιών.
* Υπο-πακέτα όπως το `scipy.signal` είναι εξειδικευμένα στην επεξεργασία σημάτων.

|  |  |
| --- | --- |
| 1. scipy.signal   (scipy.signal - Documentation, n.d.) |  |

Γιατί το επιλέξαμε: Εξειδικευμένο στην ανάλυση και επεξεργασία σημάτων, παρέχει εργαλεία για φίλτρα, μετασχηματισμούς Fourier και άλλα.

Πλεονεκτήματα:

* Ευρεία γκάμα εργαλείων για σχεδίαση και ανάλυση φίλτρων.
* Λειτουργίες για τη δημιουργία και επεξεργασία σημάτων.

|  |  |
| --- | --- |
| 1. commpy   (CommPy - Documentation, n.d.) |  |

Γιατί το επιλέξαμε: Είναι μια βιβλιοθήκη σχεδιασμένη για προσομοίωση συστημάτων επικοινωνιών, καλύπτοντας διαμορφώσεις, κανάλια και κωδικοποίηση.

Πλεονεκτήματα:

* Εξειδικευμένες λειτουργίες για ψηφιακές διαμορφώσεις όπως ASK, PSK, FSK, QAM.
* Εργαλεία για την προσομοίωση θορύβου και καναλιών επικοινωνίας.
* Εύκολη χρήση και καλή τεκμηρίωση.

|  |  |
| --- | --- |
| 1. Math   (Math Module - Documentation, n.d.) |  |

Γιατί το επιλέξαμε: Παρέχει βασικές μαθηματικές συναρτήσεις που είναι απαραίτητες για την υλοποίηση των αλγορίθμων.

Πλεονεκτήματα:

* Ενσωματωμένη στη Python, δεν απαιτείται επιπλέον εγκατάσταση.
* Απλή και γρήγορη για βασικούς υπολογισμούς.

**Πώς συνέβαλαν στην επίτευξη του στόχου της εργασίας**

1. NumPy και SciPy: Συνέβαλαν στην αποδοτική επεξεργασία και ανάλυση των ψηφιακών σημάτων, επιτρέποντας τη δημιουργία και διαχείριση μεγάλων δεδομένων με υψηλή απόδοση.
2. scipy.signal: Επέτρεψε τον σχεδιασμό και την ανάλυση ψηφιακών φίλτρων, απαραίτητων για τις εργαστηριακές ασκήσεις που αφορούν στην επεξεργασία σημάτων και στην εφαρμογή φίλτρων.
3. CommPy: Παρείχε εξειδικευμένες λειτουργίες για τις ψηφιακές διαμορφώσεις που εξετάζονται στο μάθημα, διευκολύνοντας την προσομοίωση συστημάτων επικοινωνίας και την ανάλυση της απόδοσής τους.
4. math: Χρησιμοποιήθηκε για βασικές μαθηματικές λειτουργίες που δεν απαιτούσαν τις πιο βαριές δομές δεδομένων του NumPy, συμβάλλοντας στην απλότητα και αποδοτικότητα του κώδικα.

**Συμπέρασμα**

Συνολικά, οι βιβλιοθήκες αυτές συνεργάστηκαν αρμονικά για να προσφέρουν ένα πλήρες σύνολο εργαλείων που κάλυψαν όλες τις απαιτήσεις της προσομοίωσης των ψηφιακών συστημάτων στο πλαίσιο του μαθήματος. Η επιλογή τους βασίστηκε στην ευρεία αποδοχή τους από την επιστημονική κοινότητα, την πλούσια τεκμηρίωσή τους και την ικανότητά τους να ικανοποιήσουν τις συγκεκριμένες ανάγκες της εργασίας.

#### Γραφική αναπαράσταση δεδομένων

|  |  |
| --- | --- |
| Matplotlib  (Matplotlib - Documentation, n.d.) |  |

Γιατί το επιλέξαμε:

* Ευρεία Χρήση και Ωριμότητα: Το Matplotlib είναι μία από τις παλαιότερες και πιο καθιερωμένες βιβλιοθήκες γραφικών στην Python, με μεγάλη κοινότητα χρηστών και προγραμματιστών.
* Πλούσιες Δυνατότητες: Υποστηρίζει μια μεγάλη ποικιλία γραφημάτων, συμπεριλαμβανομένων των 2D και 3D γραφημάτων, χρονικών διαγραμμάτων, φασμάτων, διαγραμμάτων αστερισμών, histograms, stem plots κ.λπ.
* Ευκολία Χρήσης: Παρέχει μια απλή και κατανοητή διεπαφή (PyPlot), που επιτρέπει τη γρήγορη δημιουργία γραφημάτων με σύνταξη παρόμοια με το MATLAB.
* Διαδραστικότητα: Αν και είναι κυρίως σχεδιασμένο για στατικά γραφήματα, μπορεί να υποστηρίξει βασική διαδραστικότητα και animation, τα οποία είναι επαρκή για τις ανάγκες του έργου.
* Συμβατότητα: Ενσωματώνεται άριστα με άλλες βιβλιοθήκες όπως NumPy και SciPy, διευκολύνοντας την απευθείας απεικόνιση δεδομένων από πίνακες και αποτελέσματα υπολογισμών.
* Τεκμηρίωση και Πόροι: Διαθέτει εκτενή τεκμηρίωση και μεγάλο αριθμό παραδειγμάτων, που διευκολύνουν την εκμάθηση και την επίλυση προβλημάτων.

Πλεονεκτήματα:

* Προσαρμοστικότητα: Επιτρέπει την πλήρη προσαρμογή των γραφημάτων, από τα βασικά στοιχεία έως τις πιο λεπτομερείς ρυθμίσεις.
* Σταθερότητα: Ως ώριμη βιβλιοθήκη, είναι σταθερή και αξιόπιστη για παραγωγική χρήση.
* Υποστήριξη 2D και 3D Γραφημάτων: Ανάγκη για απεικόνιση δεδομένων σε τρεις διαστάσεις, όπως επιφάνειες και πλέγματα, μπορεί να καλυφθεί.

**Πώς συνέβαλε στην επίτευξη του στόχου της εργασίας**

* Οπτικοποίηση Σημάτων: Χρησιμοποιήθηκε για την απεικόνιση χρονικών διαγραμμάτων των σημάτων, επιτρέποντας στους χρήστες να δουν την εξέλιξη του σήματος στον χρόνο.
* Ανάλυση Φάσματος: Διευκόλυνε την δημιουργία γραφημάτων φασματικής πυκνότητας ισχύος, βοηθώντας στην κατανόηση των φασματικών χαρακτηριστικών των σημάτων.
* Διαγράμματα Αστερισμών: Επιτρέπει την απεικόνιση των συμβόλων σε διαμορφώσεις όπως QAM και PSK, βοηθώντας στην ανάλυση της απόδοσης των διαμορφώσεων.
* Δυνατότητα Διαδραστικότητας: Σε συνδυασμό με άλλες βιβλιοθήκες (όπως το ipywidgets), μπορεί να υποστηρίξει βασική διαδραστικότητα, επιτρέποντας στους χρήστες να τροποποιούν παραμέτρους και να βλέπουν τις αλλαγές στα γραφήματα σε πραγματικό χρόνο.
* Ενσωμάτωση σε Πλατφόρμες: Μπορεί να χρησιμοποιηθεί σε περιβάλλοντα όπως Jupyter Book, με πολλά Jupyter Notebooks εσωτερικά, διευκολύνοντας την παρουσίαση και την αλληλεπίδραση με τον κώδικα και τα γραφήματα.

**Συμπέρασμα**

Συνοψίζοντας, το Matplotlib καλύπτει πλήρως τις ανάγκες του έργου για γραφική αναπαράσταση δεδομένων, παρέχοντας την ευελιξία και τα εργαλεία που απαιτούνται για την οπτικοποίηση των αποτελεσμάτων των προσομοιώσεων ψηφιακών συστημάτων. Η επιλογή του βασίστηκε στην ευρεία αποδοχή του, την πλούσια τεκμηρίωση και την ικανότητά του να ικανοποιήσει τις συγκεκριμένες απαιτήσεις του μαθήματος.

#### Διαδραστικοί μέθοδοι εισαγωγής παραμέτρων.

|  |  |
| --- | --- |
| 1. IPython   (IPython - Documentation, n.d.) |  |

Γιατί το επιλέξαμε:

* Ενσωμάτωση με Jupyter Notebooks: Το IPython αποτελεί τη βάση για τα Jupyter Notebooks, παρέχοντας ένα ισχυρό διαδραστικό περιβάλλον για την εκτέλεση κώδικα Python.
* Δυνατότητες Διαδραστικότητας: Προσφέρει εντολές και μαγικές λειτουργίες που διευκολύνουν την ανάπτυξη διαδραστικών εφαρμογών.
* Υποστήριξη Πλούσιου Περιεχομένου: Επιτρέπει την ενσωμάτωση πλούσιου περιεχομένου όπως HTML, βίντεο και widgets μέσα στα notebooks.

Πλεονεκτήματα:

* Ευκολία Χρήσης: Παρέχει ένα φιλικό προς τον χρήστη περιβάλλον με δυνατότητες αυτόματης συμπλήρωσης και εύκολης εκτέλεσης κώδικα.
* Επεκτασιμότητα: Μπορεί να επεκταθεί με τη χρήση extensions και μαγικών εντολών.
* Κοινότητα και Τεκμηρίωση: Διαθέτει μεγάλη κοινότητα και πλούσια τεκμηρίωση.

|  |  |
| --- | --- |
| 1. ipywidgets   (ipywidgets - Documentation, n.d.) |  |

Γιατί το επιλέξαμε:

* Ενσωμάτωση με Jupyter Notebooks: Σχεδιασμένο ειδικά για χρήση μέσα στα notebooks, επιτρέπει την εύκολη δημιουργία διαδραστικών widgets.
* Πλούσια Συλλογή Widgets: Παρέχει μια μεγάλη ποικιλία από προκαθορισμένα widgets όπως sliders, text boxes, dropdown menus, buttons κ.λπ.
* Εύκολη Σύνδεση με Κώδικα: Επιτρέπει τη σύνδεση των widgets με συναρτήσεις Python, ώστε οι αλλαγές στις παραμέτρους να ενημερώνουν άμεσα τα αποτελέσματα και τα γραφήματα.

Πλεονεκτήματα:

* Ευκολία Χρήσης: Με απλή σύνταξη, οι χρήστες μπορούν να δημιουργήσουν διαδραστικές διεπαφές μέσα σε λίγες γραμμές κώδικα.
* Διαδραστικότητα σε Πραγματικό Χρόνο: Οι αλλαγές στις παραμέτρους αντικατοπτρίζονται άμεσα στα γραφήματα και τα αποτελέσματα, βελτιώνοντας την κατανόηση των εννοιών.
* Συμβατότητα με Matplotlib: Μπορεί να χρησιμοποιηθεί σε συνδυασμό με το Matplotlib για την ενημέρωση γραφημάτων σε πραγματικό χρόνο.

**Πώς συνέβαλαν στην επίτευξη του στόχου της εργασίας**

* Διαδραστική Επεξεργασία Παραμέτρων: Με τη χρήση των ipywidgets, οι φοιτητές μπορούν να τροποποιούν παραμέτρους των σημάτων και των συστημάτων, όπως το Eb/No, το roll-off factor, το επίπεδο διαμόρφωσης κ.λπ., και να βλέπουν άμεσα τις επιπτώσεις στα αποτελέσματα.
* Οπτικοποίηση Αποτελεσμάτων σε Πραγματικό Χρόνο: Συνδυάζοντας τα widgets με το Matplotlib, τα γραφήματα ενημερώνονται δυναμικά, βοηθώντας τους φοιτητές να κατανοήσουν βαθύτερα τη συμπεριφορά των συστημάτων υπό διαφορετικές συνθήκες.
* Ενίσχυση της Μάθησης: Η δυνατότητα αλληλεπίδρασης με τις παραμέτρους και η άμεση οπτικοποίηση των αποτελεσμάτων ενισχύει την κατανόηση και τη διατήρηση των γνώσεων.
* Ευκολία Ανάπτυξης: Η απλή σύνταξη και η άμεση ενσωμάτωση με το υπάρχον περιβάλλον εργασίας διευκολύνει τους εκπαιδευτές στην ανάπτυξη εκπαιδευτικού υλικού και προσομοιώσεων.
* Συνεργατικότητα: Τα Jupyter Notebooks με ενσωματωμένα widgets μπορούν να διαμοιραστούν εύκολα μεταξύ των φοιτητών, επιτρέποντας την κοινή χρήση και την συνεργατική μάθηση.
* Μείωση Απαιτήσεων Εγκατάστασης: Δεδομένου ότι τα εργαλεία λειτουργούν μέσα στα Jupyter Notebooks, οι φοιτητές δεν χρειάζεται να εγκαταστήσουν πρόσθετο λογισμικό ή να ρυθμίσουν περίπλοκα περιβάλλοντα.

**Συμπέρασμα**

Η επιλογή των IPython και ipywidgets βασίστηκε στην ικανότητά τους να παρέχουν ένα ισχυρό, ευέλικτο και εύχρηστο περιβάλλον για τη διαδραστική εισαγωγή παραμέτρων και την άμεση οπτικοποίηση των αποτελεσμάτων. Με την ενσωμάτωσή τους στο ήδη γνωστό περιβάλλον των Jupyter Notebooks, διευκολύνουν την εκπαιδευτική διαδικασία και ενισχύουν την ενεργή συμμετοχή των φοιτητών στην μάθηση των ψηφιακών συστημάτων επικοινωνίας.

Κεφάλαιο 3

# Μεθοδολογία, Σχεδίαση και Υλοποίηση

## 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.

## Μέθοδος υλοποίησης του συνολικού project

Η παρούσα εργασία περιγράφει αναλυτικά τα πρώτα στάδια υλοποίησης της διαδικτυακής εφαρμογής, η οποία αναπτύχθηκε σε συνεργασία με τον Λάμπρο Φραγκουλόπουλο. Ειδικότερα, στο παρόν τεύχος παρατίθεται η ανάλυση των βημάτων 1-9 και , τα οποία αποτελούν το θεμέλιο για την ανάπτυξη του συνολικού έργου. Για περαιτέρω πληροφορίες σχετικά με τα επόμενα βήματα και την εξέλιξη του έργου, ο αναγνώστης παραπέμπεται στην αντίστοιχη εργασία του Λάμπρου Φραγκουλόπουλου [Εργασία Λάμπρου].

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.

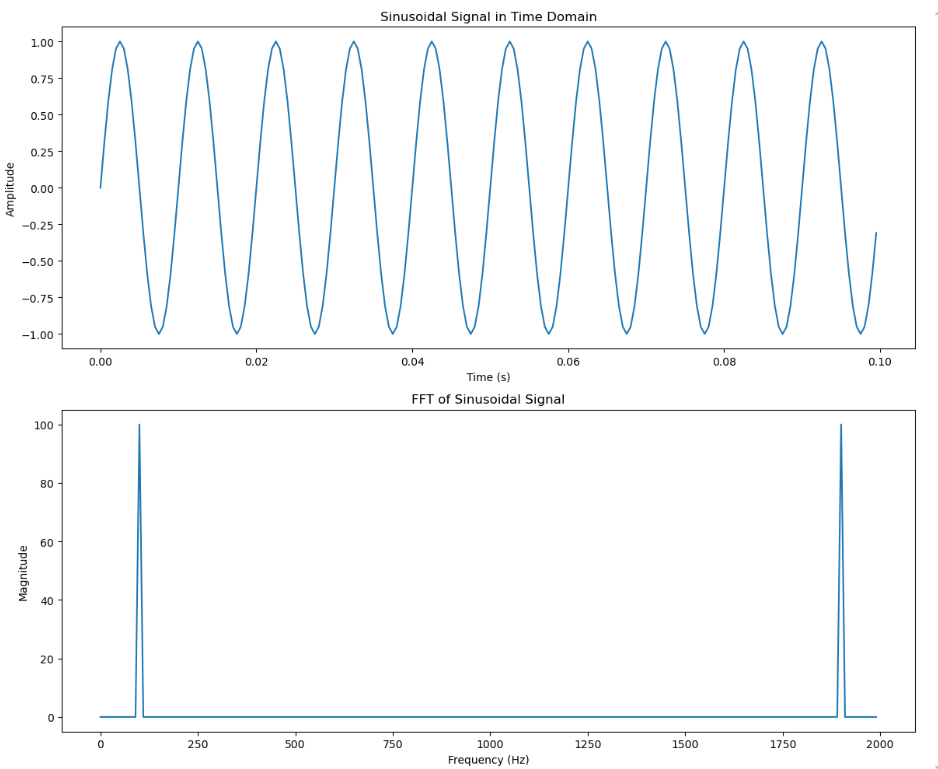
## Υλοποίηση διαδραστικών στοιχείων

### Chapter 1: Digital Signal Processing in Telecommunications

#### Δημιουργία & Οπτικοποίηση σημάτων:

Παραγωγή σημάτων για την προσομοίωση συστημάτων ψηφιακών επικοινωνιών και δημιουργία γραφημάτων για να απεικονιστούν τα χρονικά και φασματικά διαγράμματα των σημάτων.

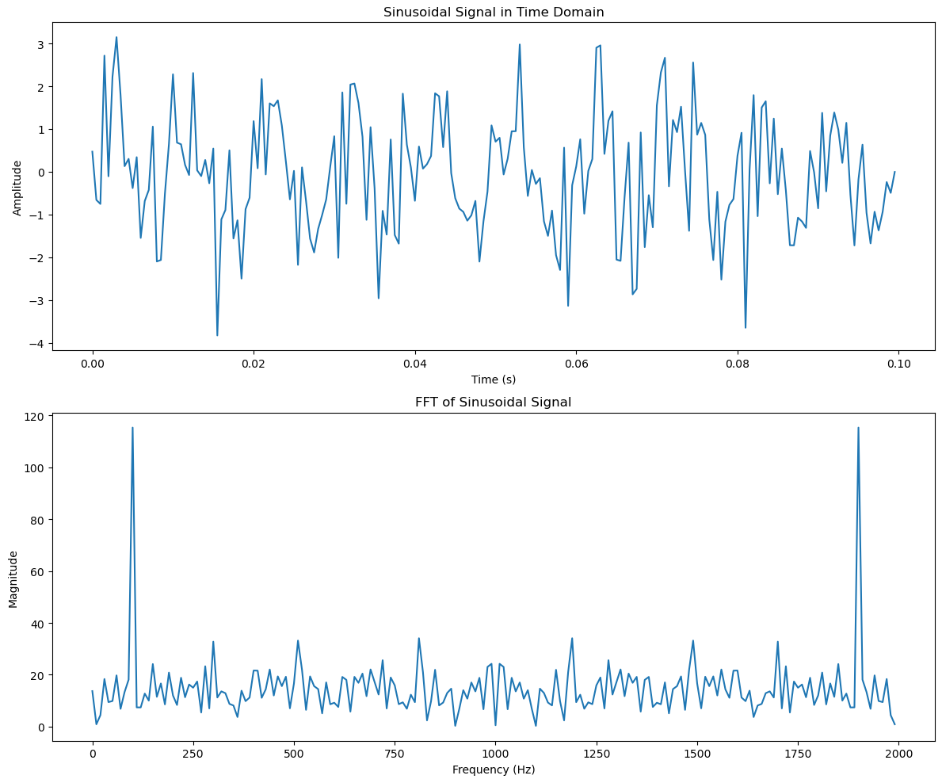
**Κώδικας – Chapter 1 – Signal Creation & Visualization**



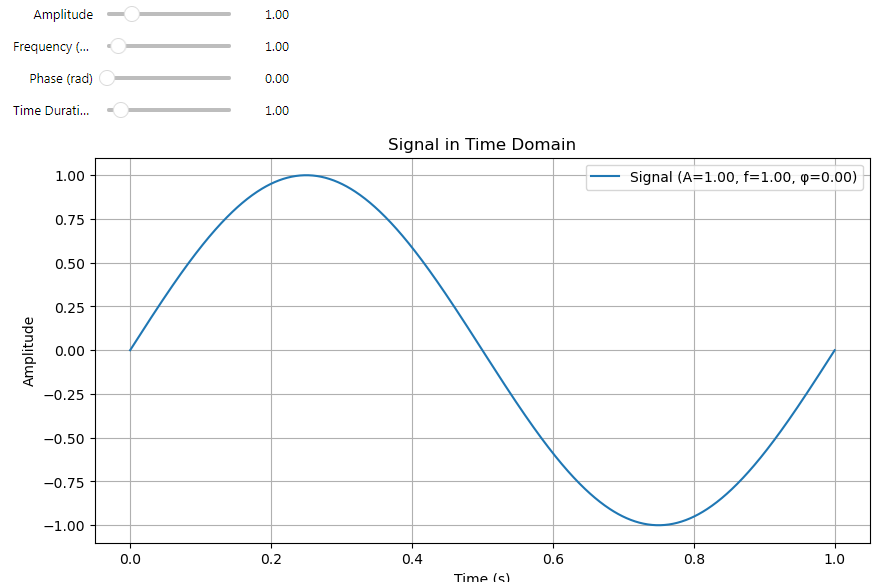
#### Προσθήκη θορύβου:

Δυνατότητα προσθήκης θορύβου στο σήμα για παρακολούθηση των επιδράσεών του στην απόδοση της επικοινωνίας.

**Κώδικας – Chapter 1 – Add noise**



#### Διαδραστική επεξεργασία μεταβλητών σήματος:

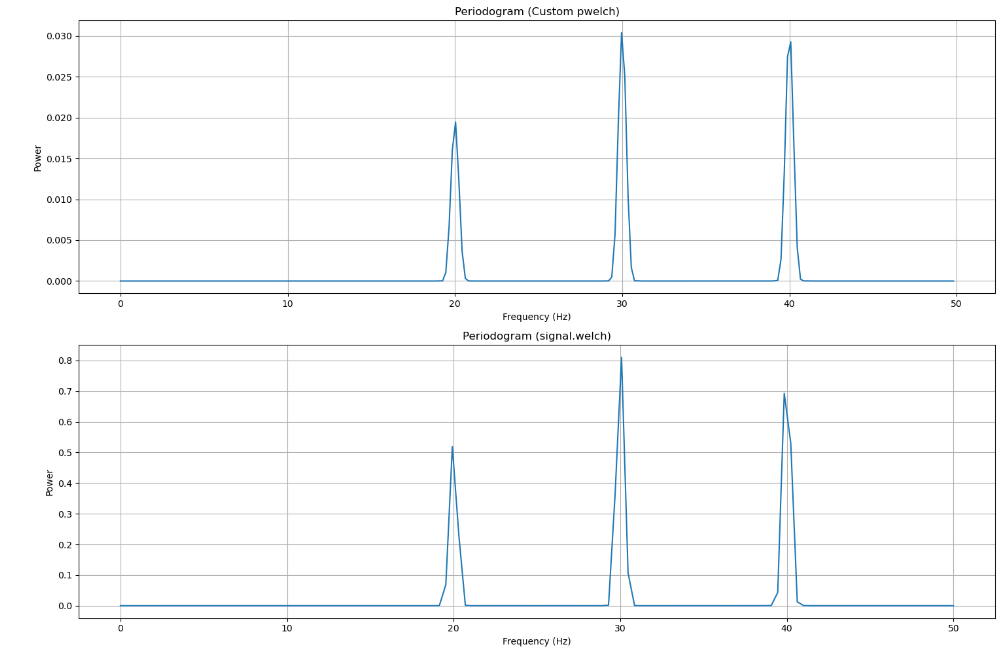
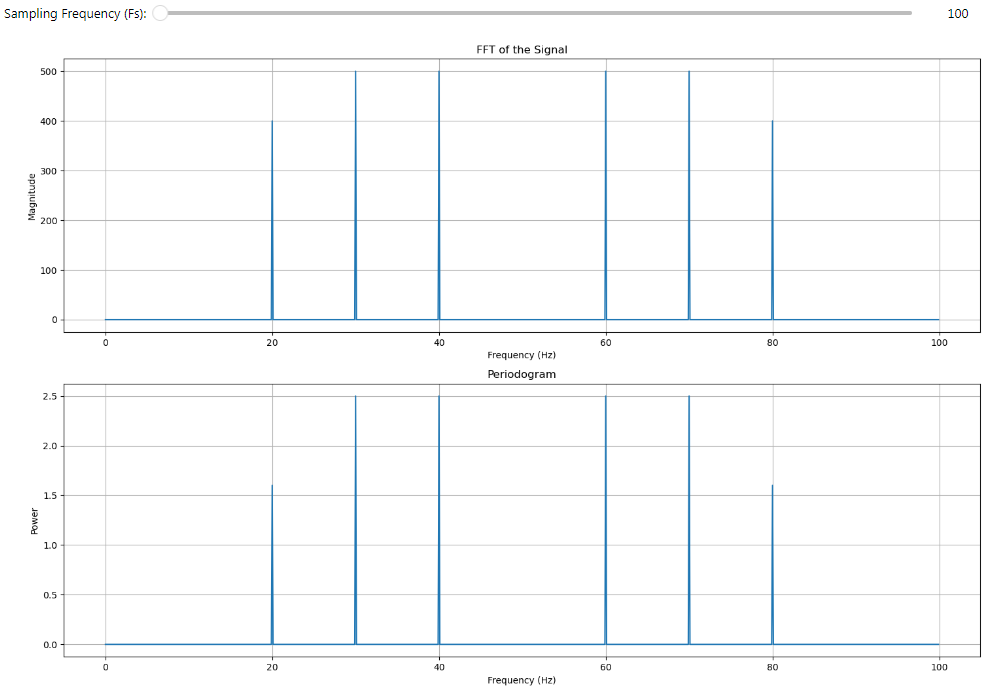
Με τη χρήση διαδραστικών εργαλίων, να μπορούν οι χρήστες να τροποποιούν τις παραμέτρους του σήματος και να παρατηρούν σε πραγματικό χρόνο τις αλλαγές.

**Κώδικας – Chapter 1 – Interactive Signal**

#### Ανάλυση Φάσματος:

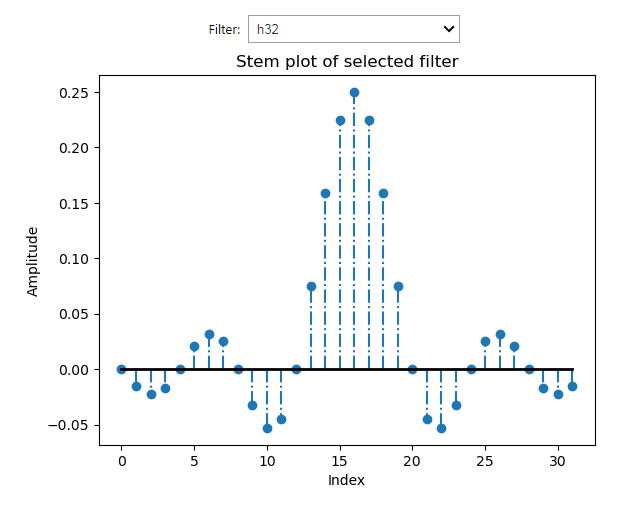
Δημιουργία γραφημάτων φασματικής ανάλυσης.

**Κώδικας – Chapter 1 - Spectrum Analysis**



#### Απεικόνιση Φίλτρου (Filter Visualization):

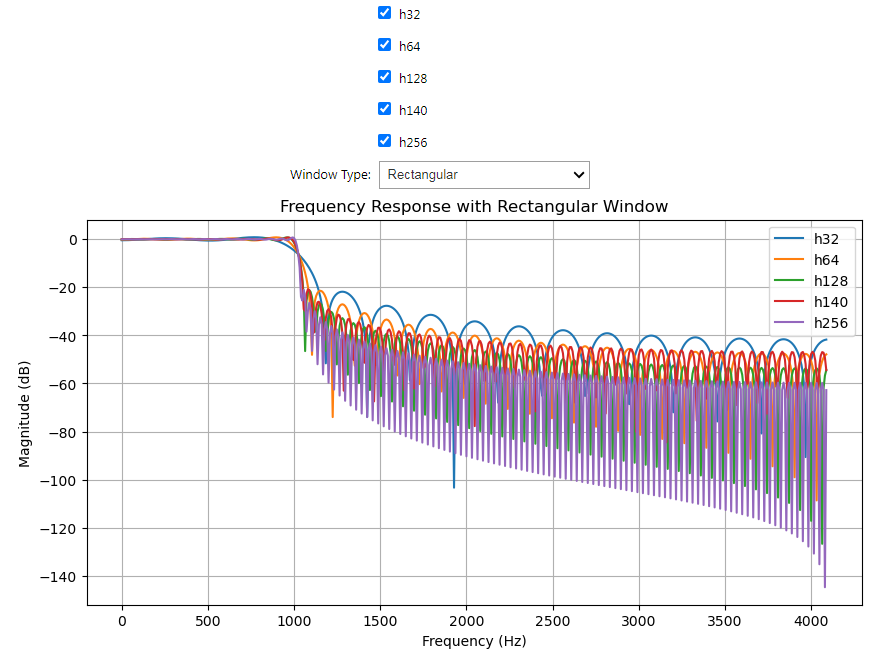
Δυνατότητα επιλογής διαφορετικών φίλτρων και οπτική αναπαράσταση της απόκρισης τους με μορφή stem plot.



**Κώδικας – Chapter 1 - Filter Visualization**

#### Ανάλυση Συχνοτήτων Φίλτρων (Filter Frequency Response):

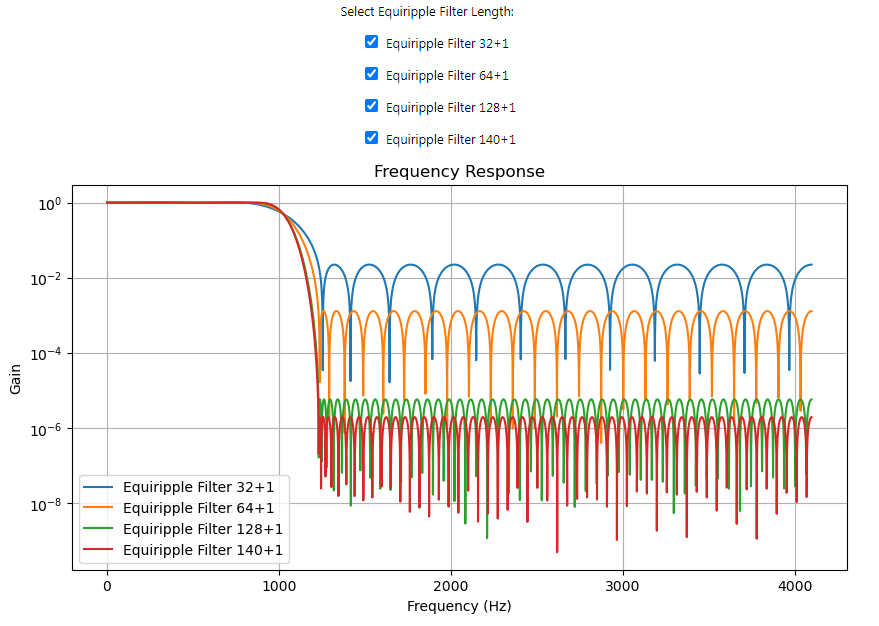
Διαδραστική απεικόνιση της φασματικής απόκρισης φίλτρων για διαφορετικά μήκη φίλτρων και παραθύρων.



**Κώδικας – Chapter 1 - Filter Frequency Response**

#### Ανάλυση Equiripple Φίλτρων:

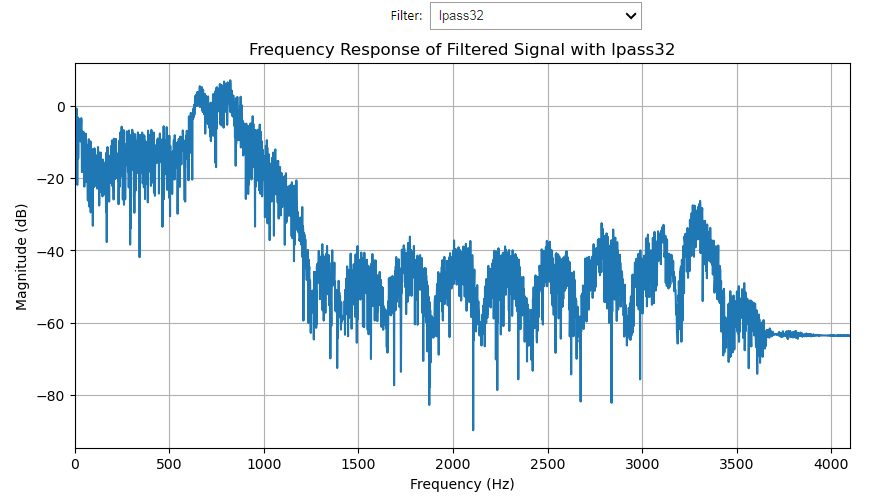
Δυνατότητα επιλογής equiripple φίλτρων διαφορετικών μηκών και σύγκριση της φασματικής τους απόκρισης.



**Κώδικας – Chapter 1 - Equiripple Filter Frequency Response**

#### Εφαρμογή Χαμηλοπερατού Φίλτρου (Low Pass Filter Application):

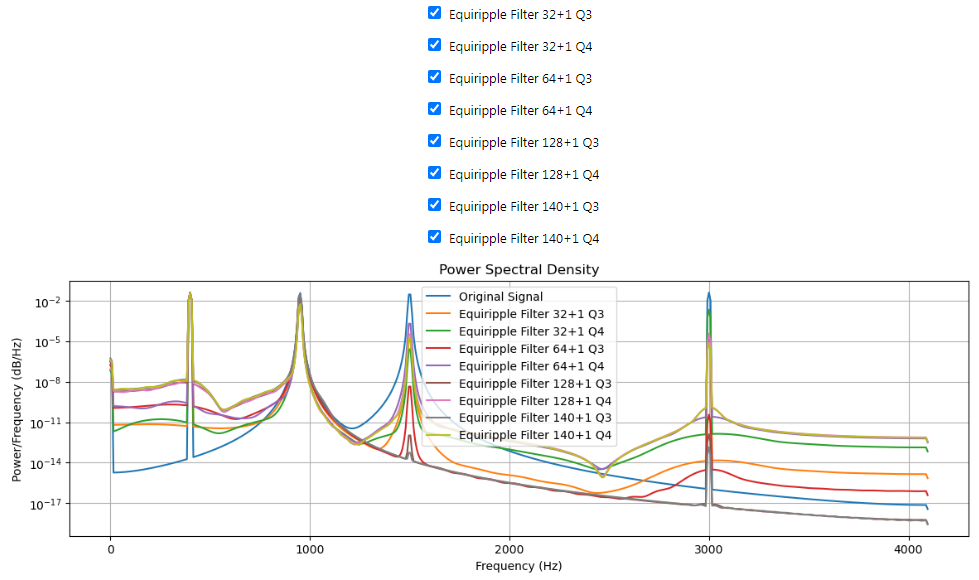
Εφαρμογή φίλτρων χαμηλής διέλευσης και απεικόνιση της απόκρισης συχνοτήτων του φιλτραρισμένου σήματος.



**Κώδικας – Chapter 1 - Low Pass Filter Application**

#### Φασματική Πυκνότητα Ισχύος (Power Spectral Density):

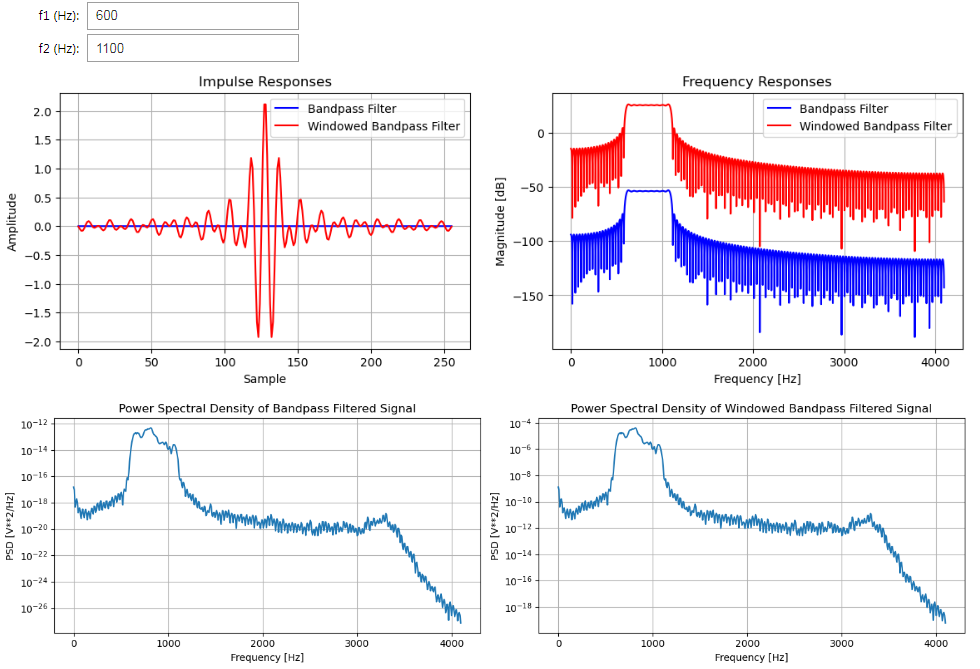
Διαδραστική σύγκριση φίλτρων equiripple και ανάλυση της φασματικής πυκνότητας ισχύος των φιλτραρισμένων σημάτων.



**Κώδικας – Chapter 1 - Power Spectral Density**

#### Ανάλυση Απόκρισης Bandpass Φίλτρων:

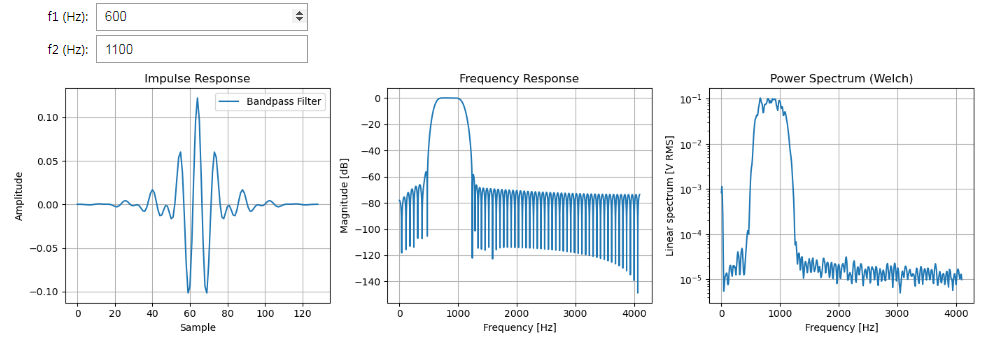
Εφαρμογή φίλτρων bandpass και οπτική απεικόνιση της απόκρισης τους τόσο στην συχνότητα όσο και στην φασματική πυκνότητα του φιλτραρισμένου σήματος.



**Κώδικας – Chapter 1 - Ανάλυση Απόκρισης Bandpass Φίλτρων**

#### Εφαρμογή Parks-McClellan Φίλτρου:

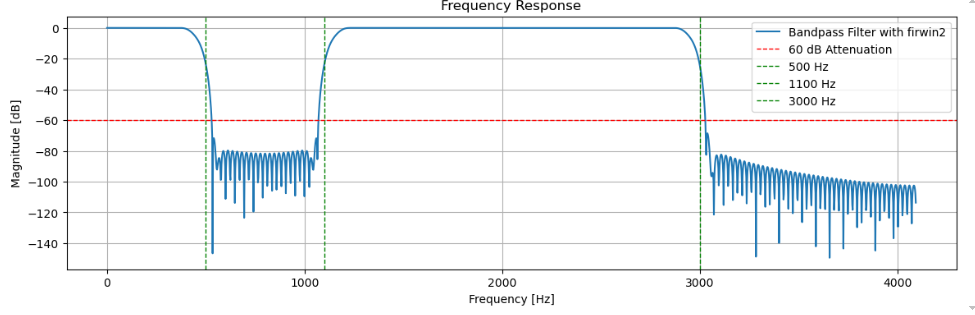
Σχεδιασμός φίλτρου bandpass με τη μέθοδο Parks-McClellan για επίτευξη συγκεκριμένης εξασθένισης στα stop bands.



**Κώδικας – Chapter 1 - Εφαρμογή Parks-McClellan Φίλτρου**

#### Εφαρμογή Φίλτρου με 2 passbands:

Σχεδιασμός και υλοποίηση φίλτρου με δύο passbands στο MATLAB, με απεικόνιση της φασματικής του απόκρισης.

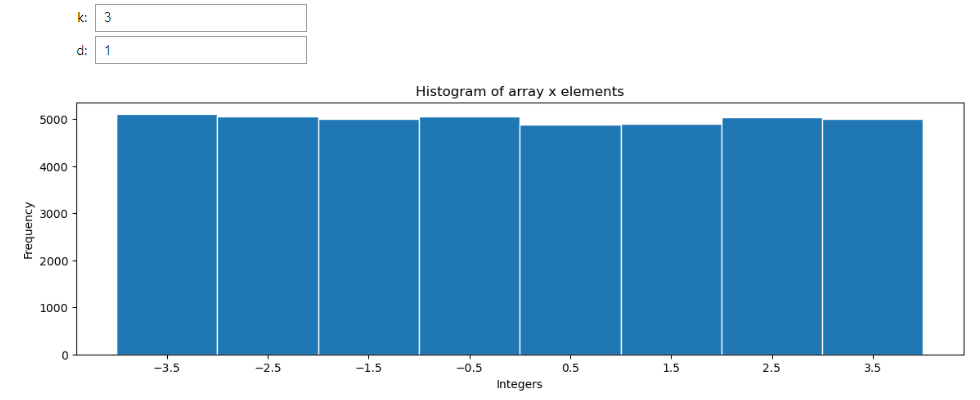


**Κώδικας – Chapter 1 - Εφαρμογή Φίλτρου με 2 passbands**

### Chapter 3: Optimal digital detection – Matched filters

#### Δημιουργία τυχαίων σημάτων:

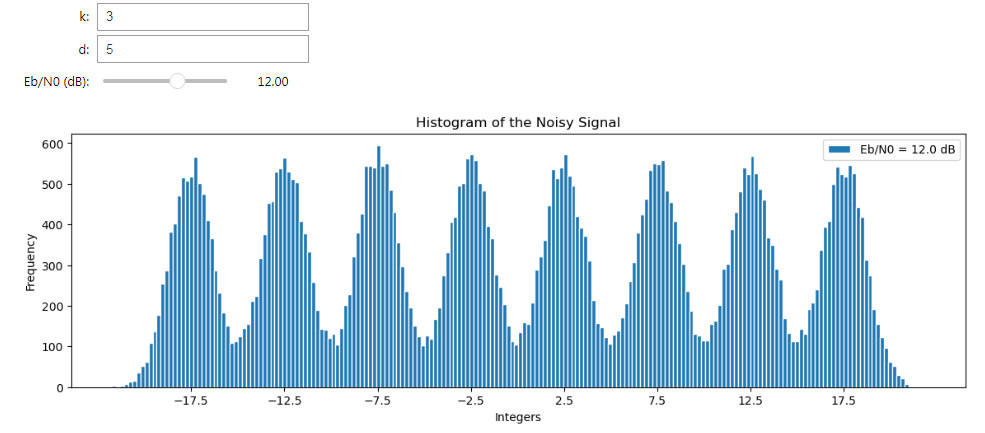
Δημιουργία τυχαίων σημάτων και παρουσίαση του ιστογράμματός τους.



**Κώδικας – Chapter 3 - Δημιουργία τυχαίων σημάτων**

#### Εφαρμογή θορύβου:

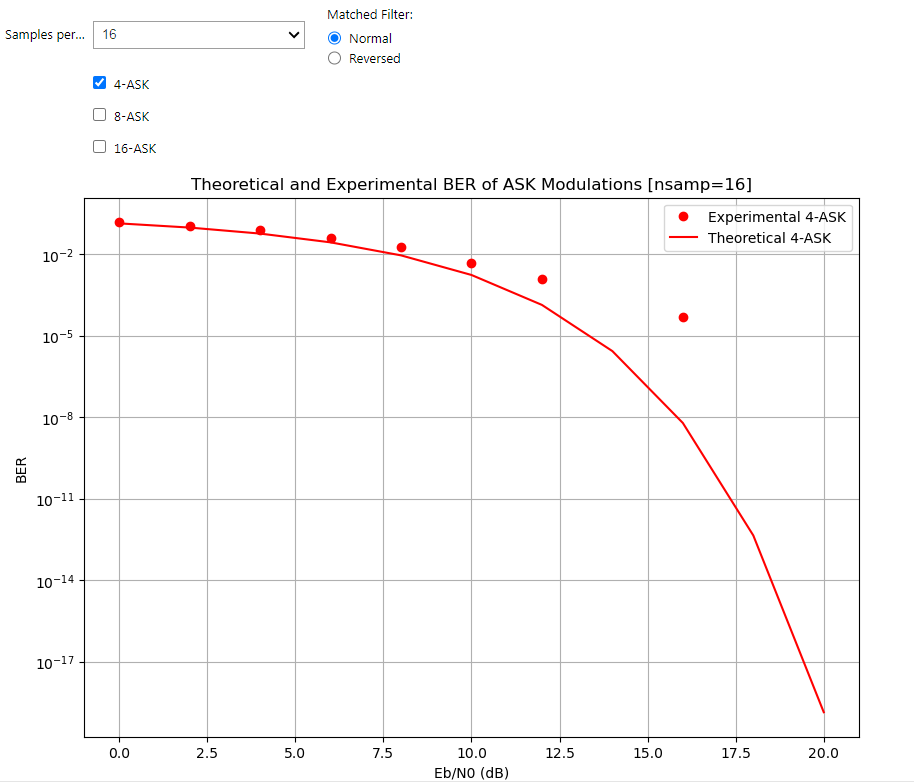
Επίδραση του θορύβου στο σήμα με τη ρύθμιση του λόγου Eb/No μέσω ενός διαδραστικού slider και τον υπολογισμό του ιστόγραμματος για διάφορες τιμές.



**Κώδικας – Chapter 3 - Εφαρμογή θορύβου**

#### ASK Bit Error Rate (BER) Visualization:

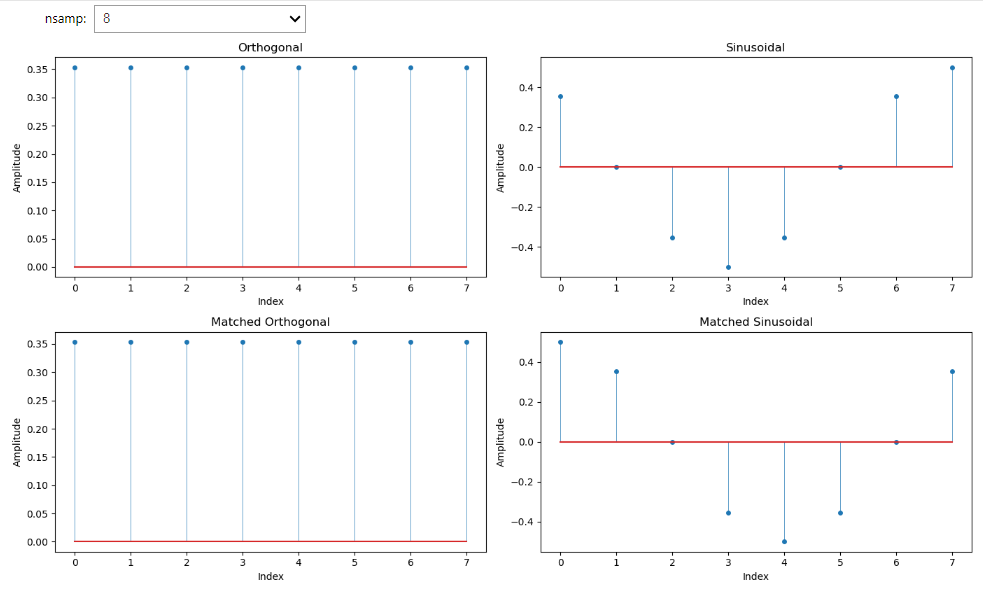
Η διαδραστική δυνατότητα σύγκρισης διαμορφώσεων Μ-ASK με επιλογές για πειραματική και θεωρητική τιμή του BER. Να μπορούν οι χρήστες να επιλέξουν μεταξύ διαφορετικών διαμορφώσεων, δειγμάτων ανά σύμβολο, και τύπου matched filter για να δουν τα αποτελέσματα σε πραγματικό χρόνο.



**Κώδικας – Chapter 3 - ASK Bit Error Rate (BER) Visualization**

#### Ανάλυση με φίλτρα:

Οπτικοποιήσεις απόκρισης φίλτρων πάνω σε ορθογώνιο και ημιτονοειδές σήμα.

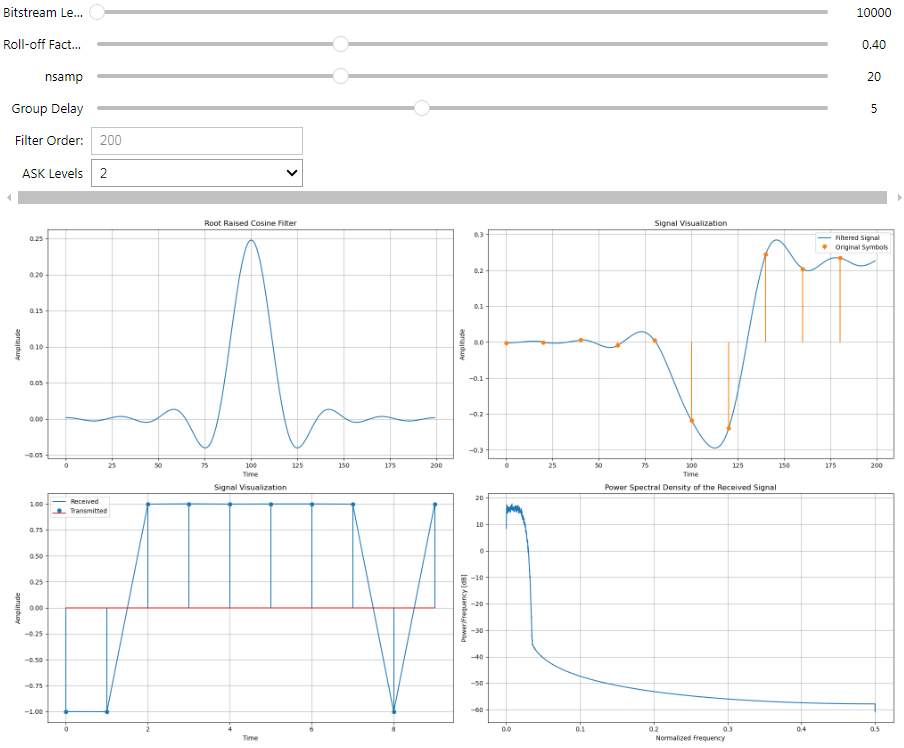


**Κώδικας – Chapter 3 - Ανάλυση με φίλτρα**

### Chapter 4: Spectral Characteristics of Digital Waveforms & Nyquist signaling

#### Ρυθμιζόμενες παράμετροι:

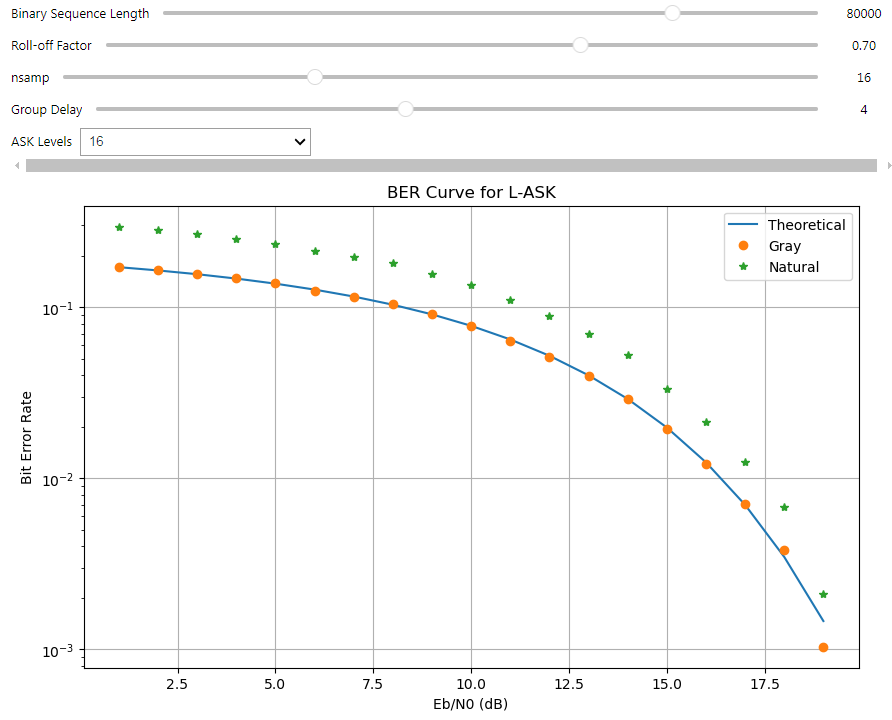
Εισαγωγή διαδραστικού τρόπου ρύθμισης παραμέτρων μήκους bitstream, roll-off factor, nsamp (δείγματα ανά σύμβολο), group delay, και σειρά του φίλτρου.



**Κώδικας – Chapter 4 - Ρυθμιζόμενες παράμετροι**

#### BER καμπύλες:

Με δυνατότητα επιλογής επιπέδου M-ASK, εμφάνισης των θεωρητικών και πειραματικών καμπυλών σε σχέση με το Eb/N0 και σύγκριση μεταξύ κωδικοποίησης Gray και Natural.



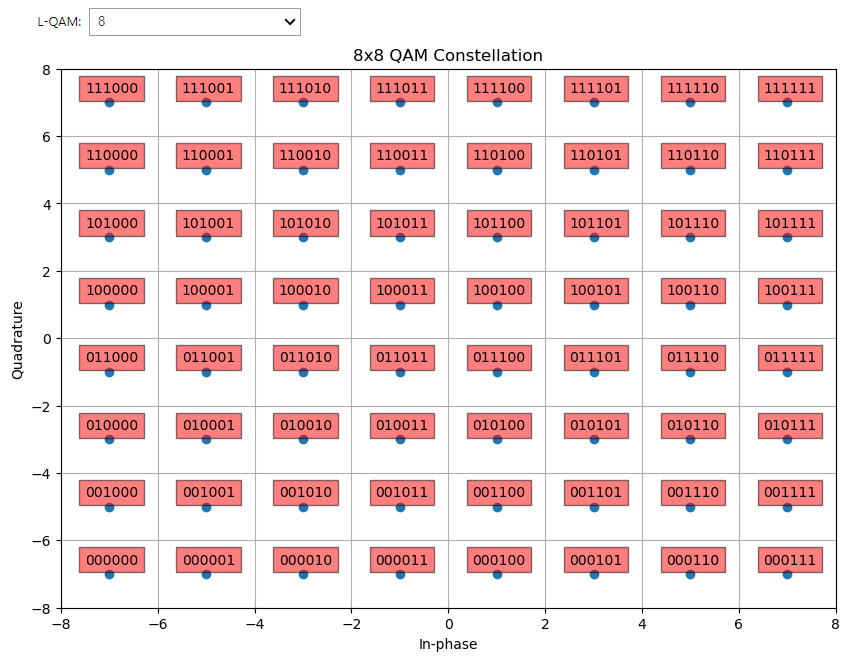
**Κώδικας – Chapter 4 - BER καμπύλες**

### Chapter 5: Digital Modulation QAM and PSK

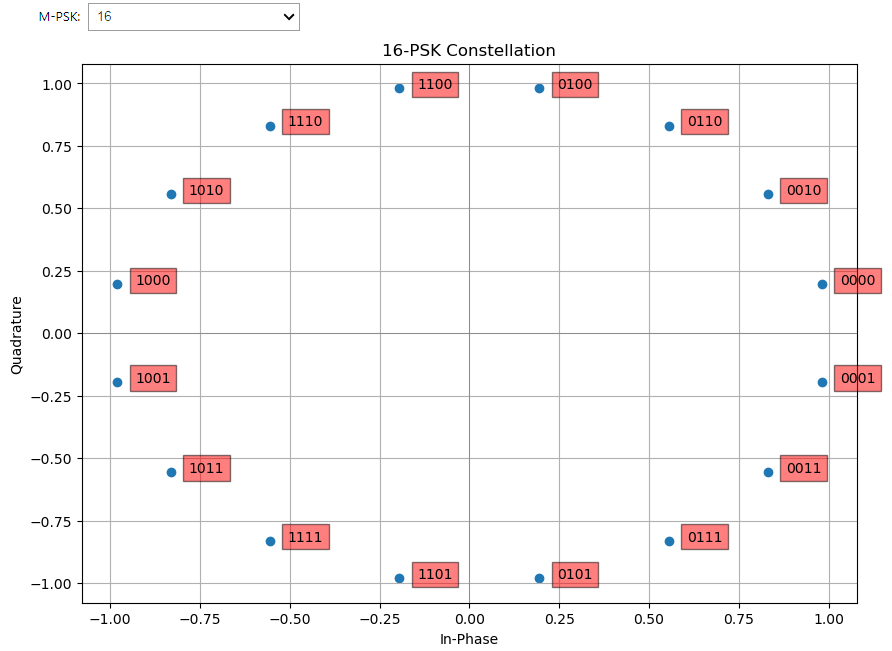
#### QAM και PSK Constellations:

Διαδραστικά διαγράμματα αστερισμών για L-QAM και M-PSK. Οι χρήστες να μπορούν να επιλέξουν το επίπεδο διαμόρφωσης και να δουν τις αντίστοιχες θέσεις των συμβόλων στον αστερισμό.

**Κώδικας – Chapter 5 - QAM Constellation**



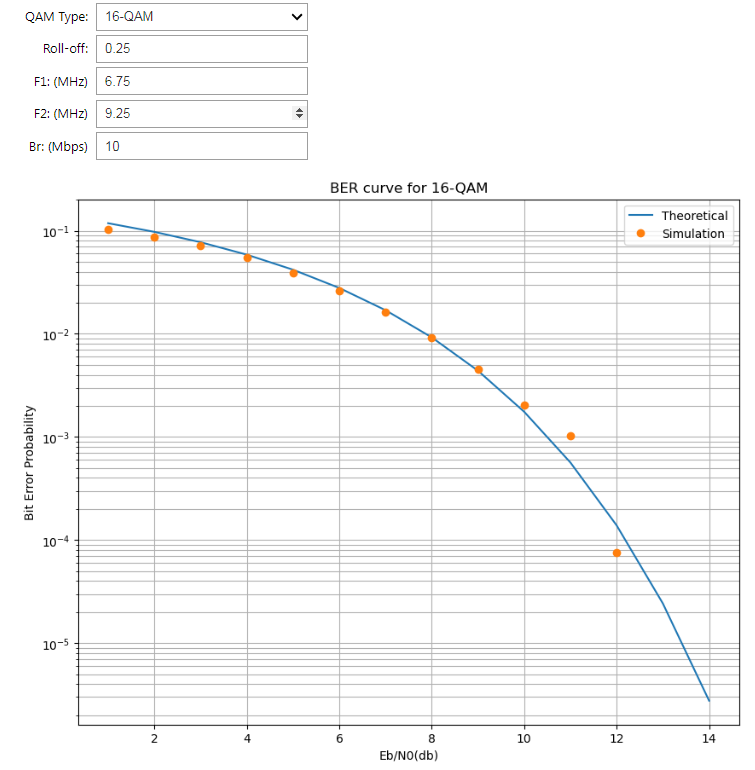
**Κώδικας – Chapter 5 - PSK Constellation**

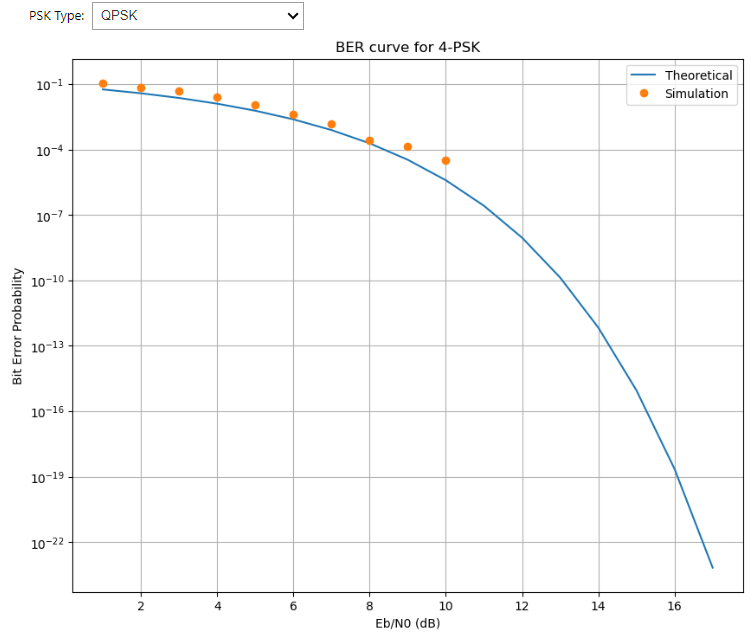


#### QAM και PSK Καμπύλες BER:

Δημιουργία γραφημάτων για τις καμπύλες BER τόσο για QAM όσο και για PSK διαμορφώσεις, όπου οι χρήστες μπορούν να συγκρίνουν την θεωρητική και πειραματική καμπύλη και να ρυθμίσουν παραμέτρους όπως το roll-off, τις συχνότητες f1 και f2, και το bitrate για την ανάλυση της πιθανότητας σφάλματος.

**Κώδικας – Chapter 5 - QAM BER**



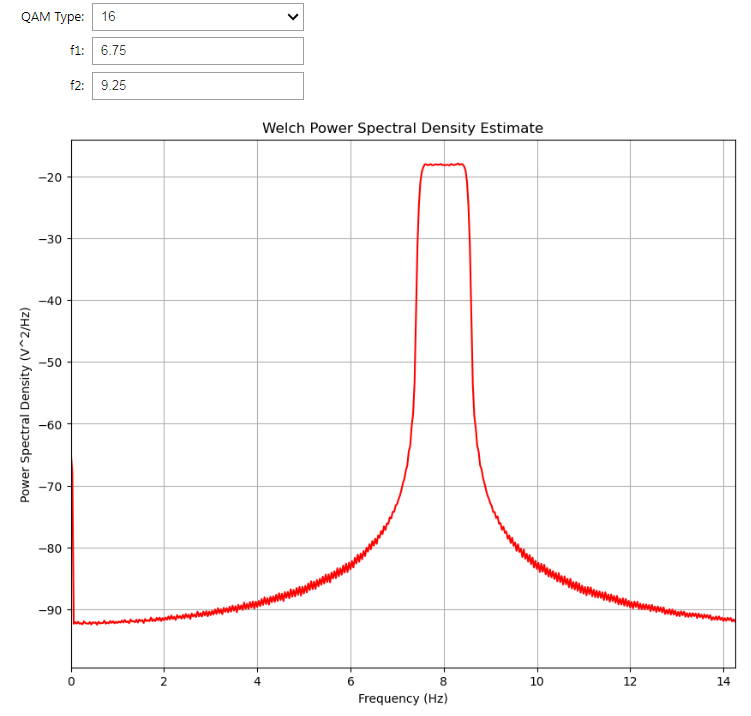


**Κώδικας – Chapter 5 - PSK BER**

#### Διαδραστική Φασματική Πυκνότητα Ισχύος:

Δυνατότητα οπτικοποίησης της φασματικής πυκνότητας ισχύος για διάφορες διαμορφώσεις QAM και συχνοτήτων f1 και f2.

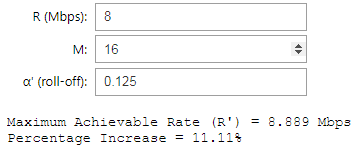
**Κώδικας – Chapter 5 - Διαδραστική Φασματική Πυκνότητα Ισχύος**



#### Διαδραστικός Υπολογισμός Παραμέτρων:

Δυνατότητα υπολογισμού παραμέτρων, με βάση τις παραμέτρους (R, M και roll-off) που επιλέγει ο χρήστης, όπως ο ρυθμός.

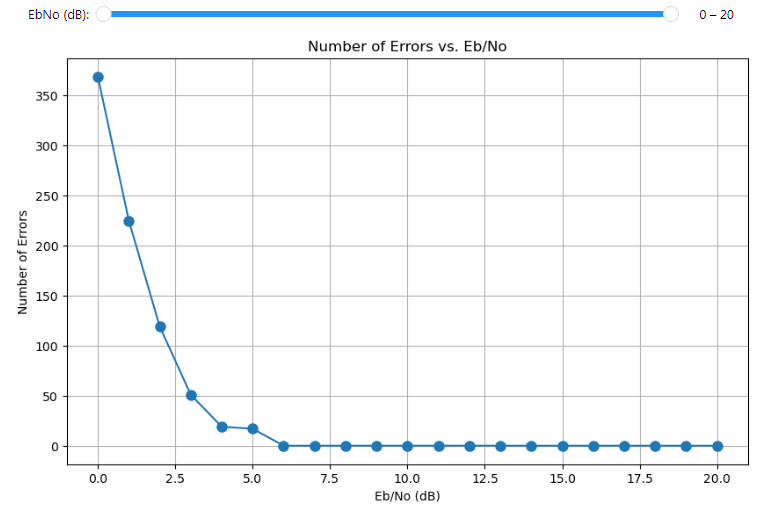
**Κώδικας – Chapter 5 - Διαδραστικός Υπολογισμός Παραμέτρων**



### Chapter 6: Digital Modulation FSK and MSK

#### Αριθμός σφαλμάτων για Eb/No:

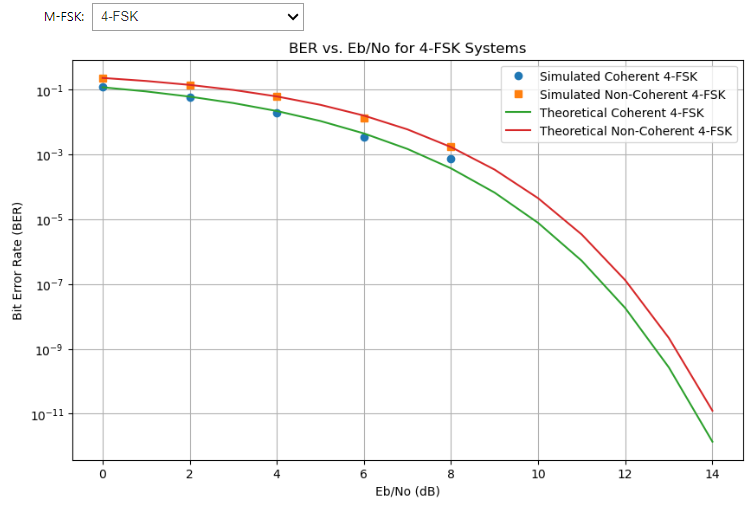
Γράφημα στο οποίο ο χρήστης ρυθμίζει ένα εύρος του Eb/No, και στη συνέχεια εμφανίζεται η καμπύλη που απεικονίζει τον αριθμό των σφαλμάτων σε σχέση με το Eb/No για συστήματα FSK.



**Κώδικας – Chapter 6 - Αριθμός σφαλμάτων για Eb/No**

#### FSK Bit Error Rate:

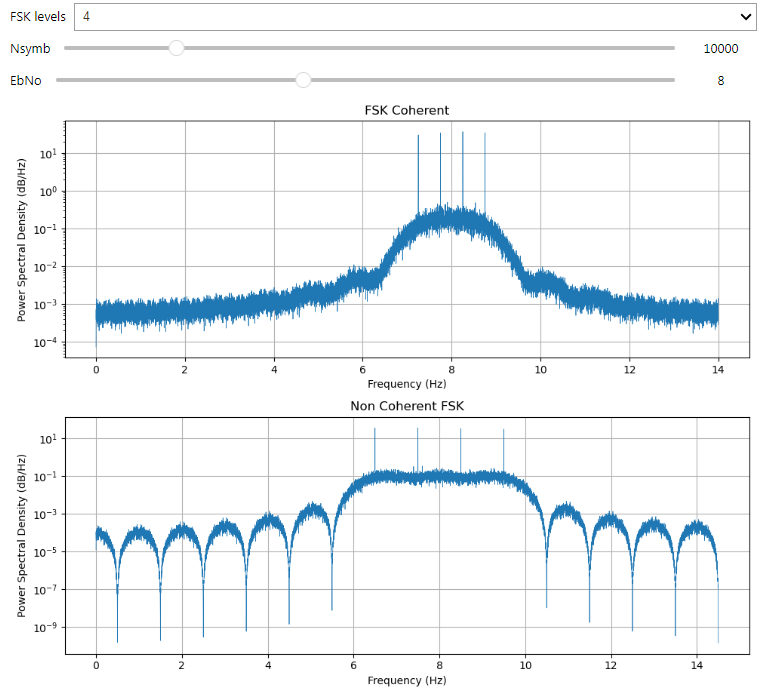
Διαδραστική σύγκριση των καμπυλών BER για coherent και non coherent συστήματα FSK. Οι χρήστες μπορούν να δουν τις καμπύλες που αντιπροσωπεύουν την θεωρητική και πειραματική απόδοση.



**Κώδικας – Chapter 6 - FSK Bit Error Rate**

#### Ανάλυση Φάσματος coherent και non coherent FSK:

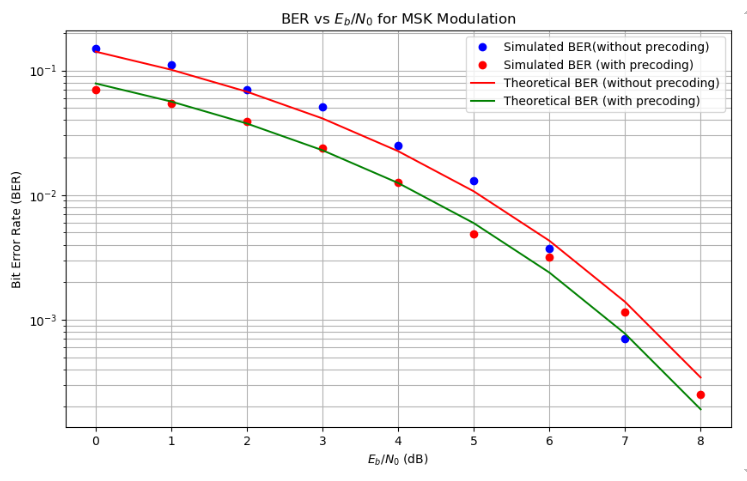
Διαδραστικό γράφημα ανάλυσης της φασματικής πυκνότητας ισχύος τόσο για coherent όσο και για non coherent FSK συστήματα, όπου οι χρήστες μπορούν να προσαρμόσουν τις παραμέτρους όπως ο αριθμός συμβόλων και το Eb/No.



**Κώδικας – Chapter 6 - Analysis coherent και non coherent FSK**

#### MSK Bit Error Rate:

Διαδραστικό γράφημα για σύγκριση μεταξύ των καμπυλών BER με και χωρίς precoding για τη διαμόρφωση MSK. Οι χρήστες μπορούν να δουν τις θεωρητικές και πειραματικές καμπύλες, συγκρίνοντας τις επιδόσεις για διαφορετικές τιμές Eb/No.



**Κώδικας – Chapter 6 - MSK Bit Error Rate**

## Custom JavaScript

Κατά την εκτέλεση των παραπάνω κομματιών κώδικα στις σελίδες του Jupyter Book, παρατηρήθηκε ότι η προβολή του Python κώδικα δεν ήταν επιθυμητή για τη βέλτιστη εμπειρία χρήστη. Προκειμένου να βελτιωθεί η αλληλεπίδραση, επιλέχθηκε η προβολή μόνο των αποτελεσμάτων και των διαδραστικών widgets. Για τον σκοπό αυτό, αναπτύχθηκε και ενσωματώθηκε κώδικας στον φάκελο ` /test-book/\_static`, ώστε κατά την εκτέλεση στις ιστοσελίδες, να αποκρύπτεται το κείμενο του κώδικα από τον χρήστη (Custom Javascript - Documentation, n.d.). Με αυτόν τον τρόπο, επιτυγχάνεται μια πιο καθαρή και άμεση παρουσίαση των αποτελεσμάτων, χωρίς περιττές πληροφορίες.

**Κώδικας – Custom JavaScript**

## Οδηγίες για τα βήματα 5-9

Λεπτομερείς οδηγίες για κάθε βήμα μπορούν να βρεθούν στο παράρτημα. Για γρήγορη περιήγηση μπορείτε να πατήσετε τους παρακάτω συνδέσμους:

* Τροποποίηση του αρχείου config.yml για την ενσωμάτωση της επιλογής binder, επιτρέποντας την εκτέλεση κώδικα σε πραγματικό χρόνο μέσω ενός εξωτερικού kernel.

**Οδηγίες - Βήμα 5**

* Δημιουργία του αρχείου requirements.txt, περιλαμβάνοντας όλες τις απαιτούμενες βιβλιοθήκες Python για την εξασφάλιση της λειτουργικότητας του kernel μέσω του MyBinder.

**Οδηγίες - Βήμα 6**

* Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για τη δημιουργία των αρχείων της ιστοσελίδας.

**Οδηγίες - Βήμα 7**

* Χρήση του GitHub Pages για την ανάπτυξη και φιλοξενία της διαδικτυακής εφαρμογής.

**Οδηγίες - Βήμα 8**

* Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για την δημοσίευση της ιστοσελίδας στο GitHub Page που δημιουργήσαμε.

**Οδηγίες - Βήμα 9**

Κεφάλαιο 4

# Συμπεράσματα

## Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζονται τα συμπεράσματα που προέκυψαν από την ανάπτυξη και τα αποτελέσματα των προηγούμενων τμημάτων κώδικα. Στη συνέχεια, αναλύονται οι διαφορές μεταξύ των προσομοιωμένων αποτελεσμάτων και των θεωρητικά αναμενόμενων γραφικών παραστάσεων, καθώς και οι χρόνοι εκτέλεσης. Επιπλέον, καταγράφονται οι προκλήσεις που προέκυψαν κατά τη διαδικασία ανάπτυξης των προσομοιώσεων και η επίδρασή τους στην ακρίβεια και την απόδοση των αποτελεσμάτων.

## Συμπεράσματα Προσομοιώσεων

Γενικά, παρατηρείται ότι ο κώδικας σε MATLAB είναι ταχύτερος σε σύγκριση με τον αντίστοιχο κώδικα σε Python, ακόμη και με τη χρήση των βιβλιοθηκών που αναφέρονται. Η διαφορά αυτή οφείλεται στο γεγονός ότι το MATLAB είναι σχεδιασμένο ειδικά για μαθηματικούς και αριθμητικούς υπολογισμούς, καθιστώντας το πιο αποδοτικό για προσομοιώσεις συστημάτων ψηφιακών επικοινωνιών. Επιπλέον, οι ενσωματωμένες βιβλιοθήκες του MATLAB, όπως το Communications Toolbox, είναι εξαιρετικά βελτιστοποιημένες για συγκεκριμένες εφαρμογές, επιτρέποντας την εκμετάλλευση των δυνατοτήτων πολυνηματικής επεξεργασίας και χρήσης GPU. Παρόλο που η Python διαθέτει ισχυρές βιβλιοθήκες, όπως η NumPy και η SciPy, συχνά απαιτούν παραπάνω βελτιστοποίηση για να επιτύχουν ανάλογες επιδόσεις.

Ωστόσο, η σύγκριση δεν είναι απόλυτα ισοδύναμη, καθώς ο κώδικας MATLAB δεν περιλαμβάνει τις διαδραστικές δυνατότητες που προστέθηκαν στην υλοποίηση με Python. Η Python προσφέρει ευελιξία και πρόσβαση σε πληθώρα βιβλιοθηκών ανοιχτού κώδικα, καθιστώντας την δημοφιλή στην ακαδημαϊκή κοινότητα, ιδιαίτερα για την ανάπτυξη διαδραστικών εφαρμογών.

Στις περισσότερες περιπτώσεις, δεδομένου ότι ο χρόνος εκτέλεσης είναι εξαιρετικά μικρός και στις δύο υλοποιήσεις, η διαφορά μπορεί να θεωρηθεί αμελητέα. Ωστόσο, στα γραφήματα του Bit Error Rate (BER) παρατηρείται σημαντική απόκλιση στον χρόνο εκτέλεσης, με τον κώδικα Python να παρουσιάζει καθυστέρηση σε σχέση με τον αντίστοιχο σε MATLAB. Αυτό οφείλεται κυρίως στην εξειδίκευση του MATLAB στους αριθμητικούς υπολογισμούς, καθώς και στην ικανότητά του να εκμεταλλεύεται καλύτερα το διαθέσιμο υλικό για απαιτητικές υπολογιστικές εργασίες.

Κεφάλαιο 5

# Μελλοντικές επεκτάσεις

## Επέκταση σε Εργαστηριακές Ασκήσεις

Με βάση τα παραπάνω κομμάτια κώδικα, στο τεύχος [Εργασία Λάμπρου] περιγράφεται πως υλοποιήθηκε η ένταξη των εργαστηριακών ασκήσεων στο Jupyter Book για τη επίτευξη της τελικής διαδικτυακής εφαρμογής.

## Εισαγωγή νέων εργαλείων

Με βάση τα εργαλεία που χρησιμοποιήσαμε στο project, μπορούμε να εξετάσουμε τις ακόλουθες επεκτάσεις και προσθήκες στο μέλλον:

1. **Plotly ή Bokeh**: Ενσωμάτωση προηγμένων διαδραστικών γραφημάτων για καλύτερη οπτικοποίηση δεδομένων.
2. **Ενσωμάτωση Machine Learning**: Χρήση βιβλιοθηκών όπως το scikit-learn για προχωρημένες αναλύσεις.
3. **Ανάπτυξη Mobile Εφαρμογής**: Χρήση πλαισίων όπως το Kivy για πρόσβαση από κινητές συσκευές.
4. **Διεθνοποίηση (i18n)**: Υποστήριξη πολλαπλών γλωσσών για ευρύτερο κοινό.
5. **Αυτοματισμός με CI/CD**: Ενσωμάτωση εργαλείων όπως το GitHub Actions για συνεχή ανάπτυξη και δοκιμές.
6. **Ενσωμάτωση Βάσεων Δεδομένων**: Χρήση SQL ή NoSQL βάσεων για αποθήκευση και διαχείριση δεδομένων.
7. **Cloud Deployment**: Φιλοξενία της εφαρμογής σε πλατφόρμες όπως AWS ή Azure για καλύτερη κλιμάκωση.
8. **Ασφάλεια και Ταυτοποίηση**: Προσθήκη μηχανισμών πιστοποίησης χρηστών για προστασία δεδομένων.
9. **Ενσωμάτωση Realtime Συνεργασίας**: Χρήση εργαλείων όπως το Google Colab για συνεργατική εργασία.
10. **Προηγμένες Οπτικοποιήσεις 3D**: Χρήση βιβλιοθηκών όπως το VisPy για τρισδιάστατες αναπαραστάσεις.
11. **Χρήση Docker Containers**: Για ευκολότερη ανάπτυξη και μεταφορά της εφαρμογής.

Κεφάλαιο 6

# Σύνοψη

## Στόχοι της Εργασίας

Οι στόχοι της παρούσας εργασίας επικεντρώνονται στην αναζήτηση, αξιολόγηση, επιλογή και χρήση κατάλληλων εργαλείων που θα υποστηρίξουν τη δημιουργία μιας διαδραστικής ιστοσελίδας για την προσομοίωση συστημάτων ψηφιακών επικοινωνιών. Η πλατφόρμα αυτή στοχεύει να προσφέρει στους φοιτητές τη δυνατότητα να εξερευνούν και να κατανοούν καλύτερα τις αρχές των ψηφιακών επικοινωνιών, μέσα από πρακτικές εφαρμογές και πειραματισμό.

Ένας βασικός στόχος της εργασίας είναι να εντοπιστούν εργαλεία που προσφέρουν επαρκή διαδραστικότητα και ευκολία χρήσης, διατηρώντας παράλληλα την ευελιξία που απαιτείται για την προσομοίωση και ανάλυση συστημάτων σε πραγματικό χρόνο. Η έρευνα επικεντρώνεται σε εργαλεία ανοιχτού κώδικα, τα οποία επιτρέπουν την ενσωμάτωση διαδραστικών στοιχείων σε εκπαιδευτικά περιβάλλοντα.

Η γλώσσα προγραμματισμού Python, λόγω της ευρείας χρήσης της στις επιστήμες των δεδομένων και της ανάλυσης, αποτελεί κεντρικό εργαλείο στην υλοποίηση των στόχων αυτών. Τέλος, απαραίτητο στοιχείο θα αποτελέσει η ενσωμάτωση kernel στην ιστοσελίδα για τη ζωντανή εκτέλεση του κώδικα Python απευθείας στον φυλλομετρητή, εξασφαλίζοντας, έτσι, για τους χρήστες, έναν εύκολο και άμεσο τρόπο αλληλεπίδρασης με τις προσομοιώσεις σε πραγματικό χρόνο.

## Περίληψη κύριων βημάτων και μεθοδολογίας

Η παρούσα εργασία περιγράφει αναλυτικά τα πρώτα στάδια υλοποίησης της διαδικτυακής εφαρμογής, η οποία αναπτύχθηκε σε συνεργασία με τον Λάμπρο Φραγκουλόπουλο. Ειδικότερα, στο παρόν τεύχος παρατίθεται η ανάλυση των βημάτων 1-9 και , τα οποία αποτελούν το θεμέλιο για την ανάπτυξη του συνολικού έργου. Για περαιτέρω πληροφορίες σχετικά με τα επόμενα βήματα και την εξέλιξη του έργου, ο αναγνώστης παραπέμπεται στην αντίστοιχη εργασία του Λάμπρου Φραγκουλόπουλου [Εργασία Λάμπρου].

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.

## Συμπεράσματα

Γενικά, παρατηρείται ότι ο κώδικας σε MATLAB είναι ταχύτερος σε σύγκριση με τον αντίστοιχο κώδικα σε Python, ακόμη και με τη χρήση των βιβλιοθηκών που αναφέρονται. Η διαφορά αυτή οφείλεται στο γεγονός ότι το MATLAB είναι σχεδιασμένο ειδικά για μαθηματικούς και αριθμητικούς υπολογισμούς, καθιστώντας το πιο αποδοτικό για προσομοιώσεις συστημάτων ψηφιακών επικοινωνιών. Επιπλέον, οι ενσωματωμένες βιβλιοθήκες του MATLAB, όπως το Communications Toolbox, είναι εξαιρετικά βελτιστοποιημένες για συγκεκριμένες εφαρμογές, επιτρέποντας την εκμετάλλευση των δυνατοτήτων πολυνηματικής επεξεργασίας και χρήσης GPU. Παρόλο που η Python διαθέτει ισχυρές βιβλιοθήκες, όπως η NumPy και η SciPy, συχνά απαιτούν παραπάνω βελτιστοποίηση για να επιτύχουν ανάλογες επιδόσεις.

Ωστόσο, η σύγκριση δεν είναι απόλυτα ισοδύναμη, καθώς ο κώδικας MATLAB δεν περιλαμβάνει τις διαδραστικές δυνατότητες που προστέθηκαν στην υλοποίηση με Python. Η Python προσφέρει ευελιξία και πρόσβαση σε πληθώρα βιβλιοθηκών ανοιχτού κώδικα, καθιστώντας την δημοφιλή στην ακαδημαϊκή κοινότητα, ιδιαίτερα για την ανάπτυξη διαδραστικών εφαρμογών.

Στις περισσότερες περιπτώσεις, δεδομένου ότι ο χρόνος εκτέλεσης είναι εξαιρετικά μικρός και στις δύο υλοποιήσεις, η διαφορά μπορεί να θεωρηθεί αμελητέα. Ωστόσο, στα γραφήματα του Bit Error Rate (BER) παρατηρείται σημαντική απόκλιση στον χρόνο εκτέλεσης, με τον κώδικα Python να παρουσιάζει καθυστέρηση σε σχέση με τον αντίστοιχο σε MATLAB. Αυτό οφείλεται κυρίως στην εξειδίκευση του MATLAB στους αριθμητικούς υπολογισμούς, καθώς και στην ικανότητά του να εκμεταλλεύεται καλύτερα το διαθέσιμο υλικό για απαιτητικές υπολογιστικές εργασίες.

# Πηγές

*Altair - Documentation*. (n.d.). Ανάκτηση από https://altair-viz.github.io/

*Bokeh - Documentation*. (n.d.). Ανάκτηση από https://docs.bokeh.org/en/latest/

*CommPy - Documentation*. (n.d.). Ανάκτηση από https://github.com/veeresht/CommPy

*Dash - Documentation*. (n.d.). Ανάκτηση από https://dash.plotly.com/

*ggplot - Documentation*. (n.d.). Ανάκτηση από http://ggplot.yhathq.com/

*GitHub - Documentation*. (n.d.). Ανάκτηση από https://docs.github.com/en

*Google Colab - Documentation*. (n.d.). Ανάκτηση από https://colab.research.google.com/notebooks/intro.ipynb

*Gradio - Documentation*. (n.d.). Ανάκτηση από https://gradio.app/docs/

*Holoviews - Documentation*. (n.d.). Ανάκτηση από https://holoviews.org/

*IPython - Documentation*. (n.d.). Ανάκτηση από https://ipython.org/documentation.html

*ipywidgets - Documentation*. (n.d.). Ανάκτηση από https://ipywidgets.readthedocs.io/en/stable/

*Jupyter Book - Binder*. (n.d.). Ανάκτηση από https://jupyterbook.org/en/stable/interactive/launchbuttons.html#add-a-launch-on-binder-button

*Jupyter Book - Documentation*. (n.d.). Ανάκτηση από https://jupyterbook.org/intro.html

*Jupyter Book – Google Colab*. (n.d.). Ανάκτηση από https://jupyterbook.org/en/stable/interactive/launchbuttons.html#add-a-launch-on-google-colab-button

*Jupyter Book - JupyterHub*. (n.d.). Ανάκτηση από https://jupyterbook.org/en/stable/interactive/launchbuttons.html#add-a-launch-on-jupyterhub-button

*Jupyter Book - Notebook Interface*. (n.d.). Ανάκτηση από https://jupyterbook.org/en/stable/interactive/launchbuttons.html#control-the-notebook-interface-that-opens

*Jupyter Book - Thebe*. (n.d.). Ανάκτηση από https://jupyterbook.org/en/stable/interactive/thebe.html

*Math Module - Documentation*. (n.d.). Ανάκτηση από https://docs.python.org/3/library/math.html

*Matplotlib - Documentation*. (n.d.). Ανάκτηση από https://matplotlib.org/stable/contents.html

*MkDocs - Documentation*. (n.d.). Ανάκτηση από (https://www.mkdocs.org/)

*NumPy - Documentation*. (n.d.). Ανάκτηση από https://numpy.org/doc/

*Pandas Visualization - Documentation*. (n.d.). Ανάκτηση από https://pandas.pydata.org/docs/user\_guide/visualization.html

*Panel - Documentation*. (n.d.). Ανάκτηση από https://panel.holoviz.org/

*Plotly - Documentation*. (n.d.). Ανάκτηση από https://plotly.com/python/

*Plotly Dash - Documentation*. (n.d.). Ανάκτηση από https://dash.plotly.com/

*PyPlot - Documentation*. (n.d.). Ανάκτηση από https://matplotlib.org/stable/api/pyplot\_summary.html

*PySDR GitHub*. (n.d.). Ανάκτηση από https://github.com/ve3wwg/pySDR

*PySimpleGUI - Documentation*. (n.d.). Ανάκτηση από https://pysimplegui.readthedocs.io/en/latest/

*PyWavelets - Documentation*. (n.d.). Ανάκτηση από https://pywavelets.readthedocs.io/en/latest/

*scikit-dsp-comm - Documentation*. (n.d.). Ανάκτηση από https://scikit-dsp-comm.readthedocs.io/en/latest/

*SciPy - Documentation*. (n.d.). Ανάκτηση από https://docs.scipy.org/doc/scipy/reference/

*scipy.signal - Documentation*. (n.d.). Ανάκτηση από https://docs.scipy.org/doc/scipy/reference/signal.html

*Seaborn - Documentation*. (n.d.). Ανάκτηση από https://seaborn.pydata.org/

*SimPy - Documentation*. (n.d.). Ανάκτηση από https://simpy.readthedocs.io/en/latest/

*Streamlit - Documentation*. (n.d.). Ανάκτηση από https://docs.streamlit.io/

*Tkinter - Documentation*. (n.d.). Ανάκτηση από https://docs.python.org/3/library/tkinter.html

*VisPy - Documentation*. (n.d.). Ανάκτηση από http://vispy.org/

*Voila - Documentation*. (n.d.). Ανάκτηση από https://voila.readthedocs.io/en/stable/

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

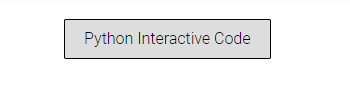
# Παράρτημα ΙΙ

## 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.

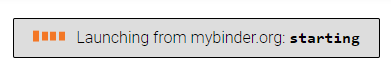
### Εκκίνηση Διαδραστικού Κώδικα

* 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.

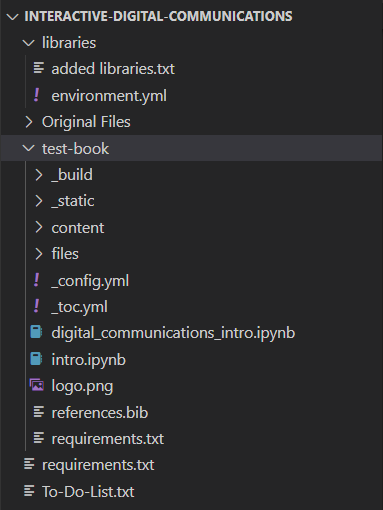


Αυτή η λειτουργία θα τρέξει όλους τους κώδικες από την αρχή, καθαρίζοντας προηγούμενες εκτελέσεις.

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

Το **Jupyter Book** είναι ένα ισχυρό εργαλείο για τη δημιουργία αλληλεπιδραστικών βιβλίων, εργαστηρίων και τεκμηρίωσης, τα οποία μπορούν να περιλαμβάνουν κώδικα, υπολογισμούς, γραφήματα και πολλά άλλα. Αυτή η τεκμηρίωση περιγράφει τα βασικά στοιχεία και τη δομή ενός Jupyter Book, προσφέροντας έναν οδηγό για τη δημιουργία περιεχομένου.

φγδ

Σχήμα 57: Απαραίτητα αρχεία του jyputer 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. Βασίζεται στο Sphinx για την τεκμηρίωση και χρησιμοποιεί MyST Markdown για τη συγγραφή κειμένων και τη διαχείριση κώδικα.

#### 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

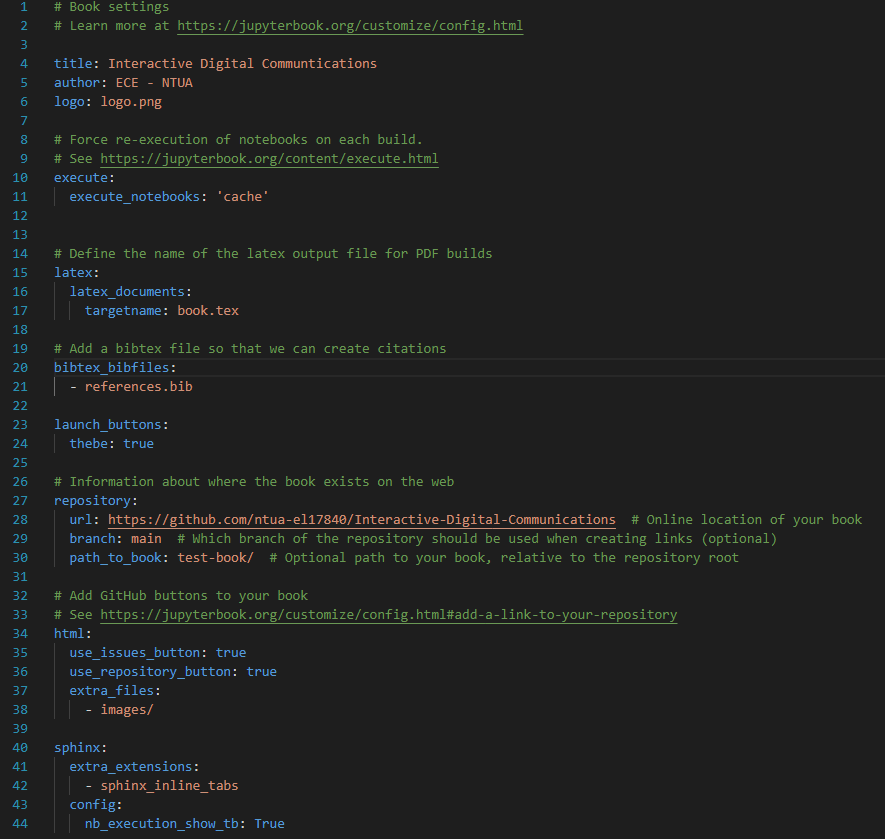
* Markdown Files (.md)
* Το MyST Markdown είναι μια επέκταση του Markdown που προσθέτει επιπλέον λειτουργίες.
* Jupyter Notebooks (.ipynb)
* Τα **Jupyter Notebooks** μπορούν να περιλαμβάνουν κείμενο, κώδικα και αποτελέσματα, όλα ενσωματωμένα σε ένα έγγραφο. Αυτά τα αρχεία επιτρέπουν τη διαδραστικότητα, καθώς ο αναγνώστης μπορεί να εκτελεί τα μπλοκ κώδικα κατά την ανάγνωση του βιβλίου.

### Ρυθμίσεις του Jupyter Book

\_config.yml

Το αρχείο **\_config.yml** καθορίζει τη γενική διαμόρφωση του βιβλίου σας. Κάποιες βασικές ρυθμίσεις περιλαμβάνουν:

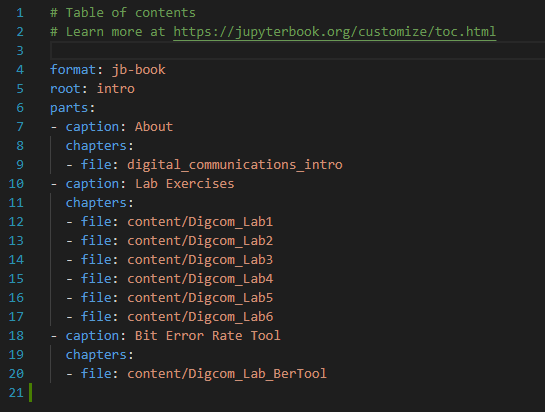
* **title**: Τίτλος του βιβλίου.
* **author**: Συγγραφέας ή ομάδα συγγραφέων.
* **logo**: Τοποθεσία του λογοτύπου που θα εμφανίζεται στο βιβλίο.
* **launch buttons**: Επιλογές για να επιτρέψετε στους χρήστες να τρέξουν το βιβλίο σε πλατφόρμες όπως το **Binder** ή το **Google Colab**.



Σχήμα 58: \_config.yml

\_toc.yml

Το αρχείο **\_toc.yml** οργανώνει το περιεχόμενο του βιβλίου σε κεφάλαια και υποκεφάλαια.

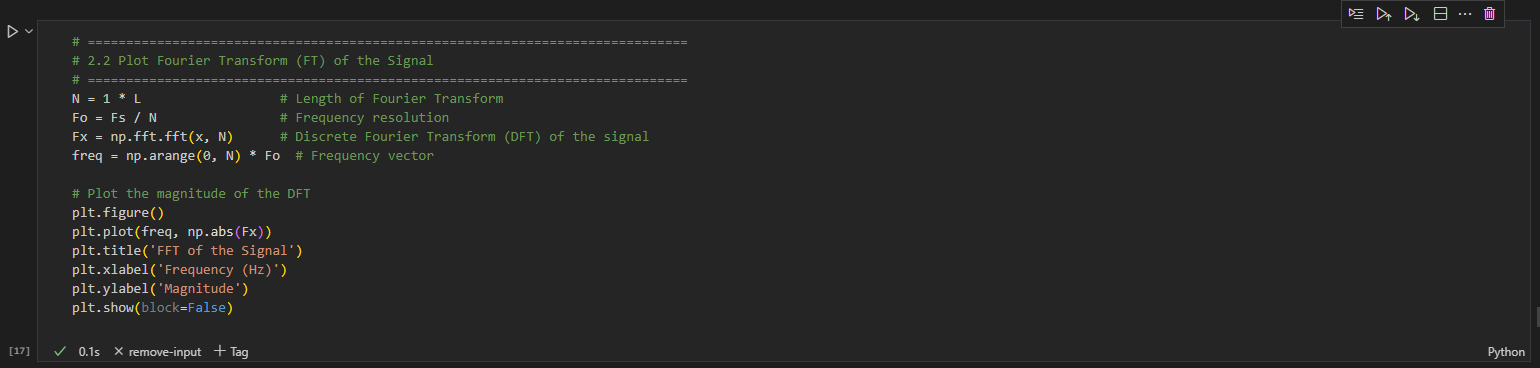


Σχήμα 59: \_toc.yml

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

### Προσθήκη Κώδικα

Το Jupyter Book υποστηρίζει την εκτέλεση κώδικα μέσω ενσωματωμένων Jupyter Notebooks ή MyST Markdown. Ο κώδικας μπορεί να τρέχει σε γλώσσες όπως Python, R, Julia, κ.λπ. Μπορείτε να δημιουργήσετε διαδραστικά κομμάτια με την παρακάτω σύνταξη:



Σχήμα 60: Εκτελέσιμο αρχείο κώδικα

### 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.

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

launch\_buttons: thebe: true

### Δημιουργία και Ανάπτυξη του Βιβλίου

Για να δημιουργήσετε το Jupyter Book:

1. Βεβαιωθείτε ότι έχετε εγκαταστήσει το Jupyter Book. Μπορείτε να το εγκαταστήσετε με την ακόλουθη εντολή:

pip install jupyter-book

1. Για να δημιουργήσετε το βιβλίο, πλοηγηθείτε στο φάκελο του βιβλίου και εκτελέστε:

jupyter-book build mybookname/

Αυτό θα δημιουργήσει το βιβλίο σας σε μορφή HTML, το οποίο μπορείτε να ανοίξετε τοπικά ή να το ανεβάσετε σε μια ιστοσελίδα.

1. Για να αναπτύξετε το βιβλίο σας στο διαδίκτυο, μπορείτε να χρησιμοποιήσετε το GitHub Pages, Netlify, ή άλλες πλατφόρμες φιλοξενίας ιστοσελίδων.

### Συμπέρασμα

Το Jupyter Book προσφέρει ένα ευέλικτο περιβάλλον για τη δημιουργία αλληλεπιδραστικών και πλούσιων σε περιεχόμενο εγγράφων, επιτρέποντας τη σύνθεση κώδικα, κειμένου και γραφημάτων. Με τη χρήση MyST Markdown, Jupyter Notebooks, και τη διαδραστικότητα που προσφέρει το Thebe, το Jupyter Book καθιστά τη δημιουργία εκπαιδευτικών εγχειριδίων και τεκμηρίωσης απλή και ισχυρή ταυτόχρονα.

Αυτή η τεκμηρίωση παρέχει έναν συνοπτικό οδηγό για τα βασικά στοιχεία και τη δομή ενός Jupyter Book, προσφέροντας τις απαραίτητες πληροφορίες για να ξεκινήσετε τη δημιουργία και τη χρήση του.

## Διαδραστικός Κώδικας

### Chapter 1: Digital Signal Processing in Telecommunications

#### Δημιουργία & Οπτικοποίηση σημάτων:

Παραγωγή σημάτων για την προσομοίωση συστημάτων ψηφιακών επικοινωνιών και δημιουργία γραφημάτων για να απεικονιστούν τα χρονικά και φασματικά διαγράμματα των σημάτων.

**Επιστροφή στην υλοποίηση**

**Κώδικας – Chapter 1 – Signal Creation & Visualization**

import numpy as np

import matplotlib.pyplot as plt

# Create a Sinusoidal Signal

Fs = 2000                  # Sampling frequency in Hz

Ts = 1 / Fs                # Sampling period in seconds

T = 0.1                    # Signal duration in seconds

t = np.arange(0, T, Ts)    # Time vector for signal

A = 1                      # Signal amplitude

x = A \* np.sin(2 \* np.pi \* 100 \* t)  # Generate sinusoidal signal

L = len(x)                 # Length of the signal

N = 1 \* L                  # Length of Fourier Transform

Fo = Fs / N                # Frequency resolution

Fx = np.fft.fft(x, N)      # DFT of the signal

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

# Plot the sinusoidal signal in time domain

plt.figure(figsize=(12, 10))

# Time-domain plot

plt.subplot(2, 1, 1)

plt.plot(t, x)

plt.title('Sinusoidal Signal in Time Domain')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

# Frequency-domain (FFT) plot

plt.subplot(2, 1, 2)

plt.plot(freq, np.abs(Fx))  # Plot only the positive frequencies

plt.title('FFT of Sinusoidal Signal')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Magnitude')

# Display the plots

plt.tight\_layout()

plt.show()

#### Προσθήκη θορύβου:

Δυνατότητα προσθήκης θορύβου στο σήμα για παρακολούθηση των επιδράσεών του στην απόδοση της επικοινωνίας.

**Επιστροφή στην υλοποίηση**

**Κώδικας – Chapter 1 – Add noise**

import numpy as np

import matplotlib.pyplot as plt

# Create a Sinusoidal Signal

Fs = 2000                  # Sampling frequency in Hz

Ts = 1 / Fs                # Sampling period in seconds

T = 0.1                    # Signal duration in seconds

t = np.arange(0, T, Ts)    # Time vector for signal

A = 1                      # Signal amplitude

x = A \* np.sin(2 \* np.pi \* 100 \* t)  # Generate sinusoidal signal

L = len(x)

# Add noise to x

rand\_n = np.random.randn(len(x))

s = x + rand\_n

N = 1 \* L                  # Length of Fourier Transform

Fo = Fs / N                # Frequency resolution

Fx = np.fft.fft(s, N)      # DFT of the signal

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

# Plot the sinusoidal signal in time domain

plt.figure(figsize=(12, 10))

# Time-domain plot

plt.subplot(2, 1, 1)

plt.plot(t, s)

plt.title('Sinusoidal Signal in Time Domain')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

# Frequency-domain (FFT) plot

plt.subplot(2, 1, 2)

plt.plot(freq, np.abs(Fx))  # Plot only the positive frequencies

plt.title('FFT of Sinusoidal Signal')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Magnitude')

# Display the plots

plt.tight\_layout()

plt.show()

#### Διαδραστική επεξεργασία μεταβλητών σήματος:

Με τη χρήση διαδραστικών εργαλίων, να μπορούν οι χρήστες να τροποποιούν τις παραμέτρους του σήματος και να παρατηρούν σε πραγματικό χρόνο τις αλλαγές.

**Κώδικας – Chapter 1 – Interactive Signal**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import interact, FloatSlider

import ipywidgets as widgets

# Initial signal parameters

def signal\_generator(amplitude, frequency, phase, time\_duration):

    # Time interval

    t = np.linspace(0, time\_duration, 1000)

    # Signal generator (sine wave)

    signal = amplitude \* np.sin(2 \* np.pi \* frequency \* t + phase)

    # Signal plot

    plt.figure(figsize=(10, 5))

    plt.plot(t, signal, label=f'Signal (A={amplitude:.2f}, f={frequency:.2f}, φ={phase:.2f})')

    plt.title('Signal in Time Domain')

    plt.xlabel('Time (s)')

    plt.ylabel('Amplitude')

    plt.grid(True)

    plt.legend()

    plt.show()

# Create interactive widgets

amplitude\_slider = FloatSlider(min=0, max=5, step=0.1, value=1, description='Amplitude', readout\_format='.2f')

frequency\_slider = FloatSlider(min=0.1, max=10, step=0.1, value=1, description='Frequency (Hz)', readout\_format='.2f')

phase\_slider = FloatSlider(min=0, max=2\*np.pi, step=0.1, value=0, description='Phase (rad)', readout\_format='.2f')

time\_slider = FloatSlider(min=0.5, max=5, step=0.1, value=1, description='Time Duration (s)', readout\_format='.2f')

# Interactive execution

interact(signal\_generator, amplitude=amplitude\_slider, frequency=frequency\_slider, phase=phase\_slider, time\_duration=time\_slider)

**Επιστροφή στην υλοποίηση**

#### Ανάλυση Φάσματος:

Δημιουργία γραφημάτων φασματικής ανάλυσης.

**Κώδικας – Chapter 1 - Spectrum Analysis**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

import ipywidgets as widgets

from ipywidgets import Layout

# Function to calculate the next power of 2 for FFT optimization

def nextpow2(i):

    n = 1

    while n < i:

        n \*= 2

    return n

# Custom implementation of pwelch function for power spectral density estimation

def pwelch(x, Fs):

    Ts = 1 / Fs  # Sampling period

    L = np.size(x) + 1  # Signal length

    T = L \* Ts  # Signal duration

    N = 2^nextpow2(L)  # Next power of 2 for FFT

    Fo = Fs / N  # Frequency resolution

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

    # Determine the window size and the number of windows

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

    if window\_size < 256:

        window\_size = 256

    windows = np.size(x) // (window\_size // 2) - 1

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

    windowed\_x = x[indexer]

    avg\_pwr = 0

    # Process each window

    for window in windowed\_x:

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

        L = np.size(window) + 1

        T = L \* Ts

        N = 2^nextpow2(L)

        Fo = Fs / N

        f = np.arange(0, N) \* Fo

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

        power = np.multiply(window\_fft, np.conj(window\_fft)) / N / L  # Power calculation

        avg\_pwr += power  # Accumulate power across windows

    avg\_pwr /= windows  # Average power over windows

    return f[np.arange(0, N // 2)], avg\_pwr[np.arange(0, N // 2)]  # Return frequency and power

# Function to update plots based on the slider value for sampling frequency

def update\_plots(sampling\_frequency):

    signal\_length = 1000  # Signal length

    sample\_period = 1 / sampling\_frequency  # Update sampling period based on slider

    time\_vector = np.arange(0, signal\_length) \* sample\_period  # Generate time vector

    # Create a composite signal with different frequencies

    signal\_data = np.sin(2 \* np.pi \* 30 \* time\_vector) + 0.8 \* np.sin(2 \* np.pi \* 80 \* (time\_vector - 2)) + np.sin(2 \* np.pi \* 60 \* time\_vector)

    # Compute custom pwelch

    custom\_frequencies, custom\_power = pwelch(signal\_data, sampling\_frequency)

    # Compute signal.welch using SciPy

    welch\_frequencies, welch\_power = signal.welch(signal\_data, fs=sampling\_frequency)

    # Compute FFT for the signal

    L = len(signal\_data)  # Signal length

    N = 1 \* L  # FFT length

    Fo = sampling\_frequency / N  # Frequency resolution

    Fx = np.fft.fft(signal\_data, N)  # Compute FFT

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

    # Compute periodogram

    power = ((Fx \* np.conj(Fx)) / (sampling\_frequency \* L)).real  # Power spectral density

    # Plot the results

    fig, axs = plt.subplots(4, 1, figsize=(15, 20))

    # Plot FFT result

    axs[0].plot(freq, np.abs(Fx))

    axs[0].set(xlabel='Frequency (Hz)', ylabel='Magnitude', title='FFT of the Signal')

    axs[0].grid()

    # Plot periodogram

    axs[1].plot(freq, power)

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

    axs[1].grid()

    # Plot custom pwelch result

    axs[2].plot(custom\_frequencies, custom\_power)

    axs[2].set(xlabel='Frequency (Hz)', ylabel='Power', title='Periodogram (Custom pwelch)')

    axs[2].grid()

    # Plot signal.welch result from SciPy

    axs[3].plot(welch\_frequencies, welch\_power)

    axs[3].set(xlabel='Frequency (Hz)', ylabel='Power', title='Periodogram (signal.welch)')

    axs[3].grid()

    plt.tight\_layout()  # Adjust layout for better fit

    plt.show()

# Create a slider for adjusting the sampling frequency

sampling\_frequency\_slider = widgets.IntSlider(

    value=500,  # Initial value

    min=100,  # Minimum frequency

    max=2000,  # Maximum frequency

    step=100,  # Step size

    description='Sampling Frequency (Fs):',

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

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

    continuous\_update=False  # Update plot only after slider adjustment

)

# Layout for slider and output

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

# Group the slider input

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

# Create interactive interface

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

# Connect slider to plot update function

output = widgets.interactive\_output(update\_plots, {'sampling\_frequency': sampling\_frequency\_slider})

# Display UI and output

display(ui, output)

**Επιστροφή στην υλοποίηση**

#### Απεικόνιση Φίλτρου (Filter Visualization):

Δυνατότητα επιλογής διαφορετικών φίλτρων και οπτική αναπαράσταση της απόκρισης τους με μορφή stem plot.

**Κώδικας – Chapter 1 - Filter Visualization**

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout

import ipywidgets as widgets

from IPython.display import display, clear\_output

import requests

import warnings

import scipy

from scipy import signal

# Fetch data from a remote source

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

response = requests.get(url)

# Convert the response text to a numpy array of floats

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

# Set sampling frequency and calculate power spectral density

Fs = 8192

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

# Define filter characteristics

N = Fs

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

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

h = np.fft.fftshift(h)

# Create filter variants of different lengths

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

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

middle = int(len(h)/2)

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

# Create a dictionary of filter variants

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 for displaying plots

output2 = widgets.Output()

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

# Function to plot the selected filter

def plot\_stem(h\_key):

    with output2:

        clear\_output(wait=True)  # Clear the previous plots

        h\_data = h\_variants[h\_key]

        x\_values = np.arange(len(h\_data))

        plt.close('all')  # Close all existing figures

        fig, ax = plt.subplots()

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

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

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

        plt.xlabel('Index')

        plt.ylabel('Amplitude')

        plt.show()

# Create a dropdown widget for filter selection

dropdown = widgets.Dropdown(options=list(h\_variants.keys()), value='h32', description='Filter:')

# Function to update the plot when the dropdown value changes

def update\_plot(change):

    plot\_stem(change['new'])

# Observe dropdown for changes

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

# Create layouts for organizing widgets

vbox\_layout = Layout(display='flex', flex\_flow='column', align\_items='center', justify\_content='space-between')

ui = widgets.HBox([dropdown], layout=Layout(align\_items='center', justify\_content='center'))

# Group the dropdown and output widget in a vertical box layout

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

# Display the final VBox

display(vbox\_final)

# Initial call to display the plot

plot\_stem(dropdown.value)

**Επιστροφή στην υλοποίηση**

#### Ανάλυση Συχνοτήτων Φίλτρων (Filter Frequency Response):

Διαδραστική απεικόνιση της φασματικής απόκρισης φίλτρων για διαφορετικά μήκη φίλτρων και παραθύρων.

**Κώδικας – Chapter 1 - Filter Frequency Response**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display, clear\_output

import scipy

from scipy import signal

import requests

import warnings

warnings.filterwarnings('ignore')

# Replace the URL with the URL of the raw content

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

response = requests.get(url)

# Split the response text by new lines and convert each line to a float

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

# Define the sampling frequency

Fs = 8192

# Calculate the power spectral density using Welch's method

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

# Define filter length and frequency response

N = Fs

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

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

h = np.fft.fftshift(h)

# Extract filter taps for different window sizes

h32 = h[N//2-16:N//2+17]

h64 = h[N//2-32:N//2+33]

h128 = h[N//2-64:N//2+65]

# Assuming H is defined and calculated as before

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

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

middle = int(len(h)/2)

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

# Extract filter taps for different window sizes

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]

# Compute frequency responses for the 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)

# Function to apply window and compute frequency responses

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

    freqs, resps = {}, {}

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

        filt\_name = f'h{filt\_size}'

        filt = eval(filt\_name)

        # Apply the selected window type

        if window\_type == 'Hamming':

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

        elif window\_type == 'Kaiser':

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

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

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

    return freqs, resps

# Initial computation with Rectangular (no window)

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

# Output widget for displaying plots

output = widgets.Output()

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

# Function to update the plot based on filter selection and window type

def update\_plot(change=None):

    window\_type = window\_type\_dropdown.value

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

    # Clear the current output and update the plot

    with output:

        clear\_output(wait=True)

        plt.figure(figsize=(10, 5))

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

            if checkboxes[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()

        plt.show()

# Create checkboxes for each filter

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

for cb in checkboxes.values():

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

# Dropdown for selecting the window type

window\_type\_dropdown = widgets.Dropdown(

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

    value='Rectangular',

    description='Window Type:',

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

)

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

# VBox layout to align items

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

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

# Combine checkboxes and dropdown into a final layout

vbox\_loader = widgets.HBox([vbox\_checkboxes])

# Combine everything into a final VBox with the layout

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

# Display the final layout

display(vbox\_final)

# Initial plot update to show default visualization

update\_plot()

**Επιστροφή στην υλοποίηση**

#### Ανάλυση Equiripple Φίλτρων:

Δυνατότητα επιλογής equiripple φίλτρων διαφορετικών μηκών και σύγκριση της φασματικής τους απόκρισης.

**Κώδικας – Chapter 1 - Equiripple Filter Frequency Response**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display, clear\_output

import scipy.signal

from scipy import signal

import requests

import warnings

warnings.filterwarnings('ignore')

# Sampling frequency

Fs = 8192

# Define different Equiripple filters with 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)

}

# Output widget to display plots

output\_plot = widgets.Output()

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

# Function to update the plot based on selected filters

def update\_plot(change=None):

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

    with output\_plot:

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

        plt.figure(figsize=(10, 5))  # Set figure size

        for filter\_name in selected\_filters:

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

            w, h = signal.freqz(filter\_coeffs, worN=8000)  # Compute frequency response

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

        plt.title('Frequency Response')

        plt.xlabel('Frequency (Hz)')

        plt.ylabel('Gain')

        plt.legend()

        plt.grid(True)

        plt.show()

# Create checkboxes for each filter

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

for cb in checkboxes:

    cb.observe(update\_plot, names='value')  # Update plot when checkbox is changed

# VBox layout to align checkboxes

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

vbox\_checkboxes = widgets.VBox([widgets.Label('Select Equiripple Filter Length:')] + checkboxes, layout=vbox\_layout)

# Layout container for checkboxes

vbox\_loader = widgets.HBox([vbox\_checkboxes])

# Final VBox layout to display checkboxes and output

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

# Display the final layout

display(vbox\_final)

# Initial plot update to show default visualization

update\_plot()

**Επιστροφή στην υλοποίηση**

#### Εφαρμογή Χαμηλοπερατού Φίλτρου (Low Pass Filter Application):

Εφαρμογή φίλτρων χαμηλής διέλευσης και απεικόνιση της απόκρισης συχνοτήτων του φιλτραρισμένου σήματος.

**Κώδικας – Chapter 1 - Low Pass Filter Application**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display, clear\_output

import scipy.signal

import requests

import warnings

warnings.filterwarnings('ignore')

# Replace the URL with the URL of the raw content

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

response = requests.get(url)

# Split the response text by new lines and convert each line to a float

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

# Sampling frequency

Fs = 8192

# Low-pass filter coefficients dictionary (lp\_filters)

lp\_filters = {

    'lpass32': scipy.signal.remez(33, [0, 0.1 \* Fs, 0.15 \* Fs, 0.5 \* Fs], [1, 0], Hz=Fs),

    'lpass64': scipy.signal.remez(65, [0, 0.1 \* Fs, 0.15 \* Fs, 0.5 \* Fs], [1, 0], Hz=Fs),

    'lpass128': scipy.signal.remez(129, [0, 0.1 \* Fs, 0.15 \* Fs, 0.5 \* Fs], [1, 0], Hz=Fs),

    'lpass140': scipy.signal.remez(141, [0, 0.1 \* Fs, 0.15 \* Fs, 0.5 \* Fs], [1, 0], Hz=Fs)

}

# Output widget for the plot

plot\_output = widgets.Output()

# Function to display the frequency response of the filtered signal

def plot\_freq\_response(filter\_name):

    with plot\_output:

        clear\_output(wait=True)

        # Apply the selected filter to the original signal

        s\_lp\_filtered = scipy.signal.convolve(s, lp\_filters[filter\_name], mode='same') / np.sum(lp\_filters[filter\_name])

        # Calculate FFT of the filtered signal

        freqs = np.fft.rfftfreq(len(s\_lp\_filtered), 1 / Fs)

        fft\_mag = np.abs(np.fft.rfft(s\_lp\_filtered))

        # Convert magnitude to dB scale

        fft\_mag\_db = 20 \* np.log10(fft\_mag)

        # Plot the frequency response

        plt.figure(figsize=(10, 5))

        plt.plot(freqs, fft\_mag\_db)

        plt.title(f'Frequency Response of Filtered Signal with {filter\_name}')

        plt.xlabel('Frequency (Hz)')

        plt.ylabel('Magnitude (dB)')

        plt.grid(True)

        plt.xlim(0, Fs / 2)

        plt.show()

# Dropdown widget to select the filter

filter\_dropdown = widgets.Dropdown(

    options=[(name, name) for name in lp\_filters.keys()],

    value='lpass32',

    description='Filter:'

)

# Function to handle dropdown selection change

def on\_filter\_change(change):

    filter\_name = change['new']

    plot\_freq\_response(filter\_name)

filter\_dropdown.observe(on\_filter\_change, names='value')

# VBox layout for alignment and presentation

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

vbox\_loader = widgets.HBox([filter\_dropdown], layout=vbox\_layout)

# Final VBox layout for the filter selection and plot display

vbox\_final = widgets.VBox([vbox\_loader, plot\_output])

# Display the final layout

display(vbox\_final)

# Initialize the plot with the default filter

on\_filter\_change({'new': filter\_dropdown.value})

**Επιστροφή στην υλοποίηση**

#### Φασματική Πυκνότητα Ισχύος (Power Spectral Density):

Διαδραστική σύγκριση φίλτρων equiripple και ανάλυση της φασματικής πυκνότητας ισχύος των φιλτραρισμένων σημάτων.

**Κώδικας – Chapter 1 - Power Spectral Density**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display, clear\_output

import scipy.signal as signal

import time

import warnings

warnings.filterwarnings('ignore')

# Sampling frequency

Fs = 8192

# Signal duration

t = np.arange(0, 1.0, 1/Fs)

# Re-define the signal with new 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 and parameters

filters5 = {

    'Equiripple Filter 32+1 Q3': signal.remez(33, [0, 0.1\*Fs, 0.15\*Fs, 0.5\*Fs], [1, 0], Hz=Fs),

    'Equiripple Filter 32+1 Q4': signal.remez(33, [0, 0.11\*Fs, 0.12\*Fs, 0.5\*Fs], [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

checkboxes = [widgets.Checkbox(value=True, description=name) for name in filters5.keys()]

# Create output widget for the plot

output\_plot = widgets.Output()

# Function to update the plot based on selected filters

def update\_plot(dummy=None):

    with output\_plot:

        clear\_output(wait=True)

        # Start a new plot

        plt.figure(figsize=(14, 4))

        # Add original signal Power Spectral Density (PSD)

        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 checkboxes:

            if cb.value:  # only plot if the checkbox is checked

                filter\_name = cb.description

                filter\_coeffs = filters5[filter\_name]

                filtered\_signal = signal.lfilter(filter\_coeffs, 1.0, s\_new)

                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')

        plt.show()

# Attach the update\_plot function to the checkboxes

for cb in checkboxes:

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

# Initial plot update

update\_plot()

# Create a VBox to hold the checkboxes

vbox\_checkboxes = widgets.VBox(checkboxes)

# Create an HBox to center the checkboxes

ui = widgets.HBox([vbox\_checkboxes], layout=Layout(justify\_content='center', align\_items='center'))

# Create a VBox to display both the checkboxes and the plot

out = widgets.VBox([ui, output\_plot])

# Display the checkboxes and the plot

display(out)

# Initialize the first plot update

update\_plot()

**Επιστροφή στην υλοποίηση**

#### Ανάλυση Απόκρισης Bandpass Φίλτρων:

Εφαρμογή φίλτρων bandpass και οπτική απεικόνιση της απόκρισης τους τόσο στην συχνότητα όσο και στην φασματική πυκνότητα του φιλτραρισμένου σήματος.

**Κώδικας – Chapter 1 - Ανάλυση Απόκρισης Bandpass Φίλτρων**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display, clear\_output

import requests

import time

from scipy.signal import kaiser\_atten, freqz, convolve, welch

import warnings

warnings.filterwarnings('ignore')

# Define your input widgets for the frequencies

f1\_input = widgets.IntText(value=600, description='f1 (Hz):', continuous\_update=False)

f2\_input = widgets.IntText(value=1100, description='f2 (Hz):', continuous\_update=False)

# Define the output area for the plots

plot\_output = widgets.Output()

def process\_signal(change):

    f1, f2 = f1\_input.value, f2\_input.value

    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 any frequency is greater than 4000

        with plot\_output:

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

            warning\_html = 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\_html)

    else:

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

        with plot\_output:

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

            plt.close('all')  # Close all existing figures to prevent memory leaks

            # Replace the URL with the URL of the raw content

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

            response = requests.get(url)

            # Split the response text by new lines and convert each line to a float

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

            Fs = 8192

            f2m1 = f2 - f1

            f2p1 = (f2 + f1) / 2

            N = 256

            Ts = 1 / Fs

            t = np.arange(-(N - 1), N, 2) \* Ts / 2

            # Bandpass filter

            hbp = 2 / Fs \* np.cos(2 \* np.pi \* f2p1 \* t) \* np.sinc(f2m1 \* t)

            hbpw = hbp \* kaiser\_atten(len(hbp), 5)

            # Compute frequency responses

            w, h = freqz(hbp, worN=8000)

            w\_w, h\_w = freqz(hbpw, worN=8000)

            fs = Fs  # Sampling frequency for normalization

            # Creating a figure and a 1x2 subplot layout

            fig, axs = plt.subplots(1, 2, figsize=(14, 4))

            # Plotting 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')

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

            axs[0].legend()

            axs[0].grid(True)

            # Plotting 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)

            # Convolution with the signal

            sima\_bp = convolve(s, hbp, mode='same')

            sima\_bpw = convolve(s, hbpw, mode='same')

            fig, axs = plt.subplots(1, 2, figsize=(14, 4))

            # Plotting the Power Spectral Density

            f, Pxx = welch(sima\_bp, Fs, nperseg=1024)

            axs[0].semilogy(f, Pxx)

            axs[0].set\_title('Power Spectral Density of Bandpass Filtered Signal')

            axs[0].set\_xlabel('Frequency [Hz]')

            axs[0].set\_ylabel('PSD [V\*\*2/Hz]')

            axs[0].grid(True)

            f, Pxx = welch(sima\_bpw, Fs, nperseg=1024)

            axs[1].semilogy(f, Pxx)

            axs[1].set\_title('Power Spectral Density 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()

            plt.show()

# Attach the event handler to the frequency inputs

f1\_input.observe(process\_signal, names='value')

f2\_input.observe(process\_signal, names='value')

# Display widgets

input\_ui = widgets.VBox([f1\_input, f2\_input])

ui = widgets.HBox([input\_ui], layout=Layout(align\_items='center'))

display(ui, plot\_output)

# Call process\_signal initially

process\_signal(None)

**Επιστροφή στην υλοποίηση**

#### Εφαρμογή Parks-McClellan Φίλτρου:

Σχεδιασμός φίλτρου bandpass με τη μέθοδο Parks-McClellan για επίτευξη συγκεκριμένης εξασθένισης στα stop bands.

**Κώδικας – Chapter 1 - Εφαρμογή Parks-McClellan Φίλτρου**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display, clear\_output

import time

from scipy.signal import firwin2, freqz, welch

import warnings

warnings.filterwarnings('ignore')

# Define the interactive widgets

f1\_input = widgets.IntText(value=600, description='f1 (Hz):', continuous\_update=False)

f2\_input = widgets.IntText(value=1100, description='f2 (Hz):', continuous\_update=False)

output\_area = widgets.Output()

# Load an example signal

s = np.random.randn(5000)  # Example signal

Fs = 8192  # Sampling frequency

# Function to update plots based on filter frequencies

def update\_plots(change):

    # Read the values from the input widgets

    f1, f2 = f1\_input.value, f2\_input.value

    if not (f1 < f2) or any(freq < 0 for freq in [f1, f2]) or any(freq > 3900 for freq in [f1, f2]):

        # Clear the output and display a warning message

        with output\_area:

            output\_area.clear\_output(wait=True)

            warning\_html = 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! Also, make sure f1 < f2!</div>",

                placeholder='',

                description='',

            )

            display(warning\_html)

    else:

        # Proceed with the plot if all frequencies are valid

        with output\_area:

            output\_area.clear\_output(wait=True)

            # Design the bandpass FIR filter

            freq11 = np.array([0, f1 \* 0.95, f1 \* 1.05, f2 \* 0.95, f2 \* 1.05, Fs / 2]) / (Fs / 2)

            gain = [0, 0, 1, 1, 0, 0]

            hbp\_pm = firwin2(129, freq11, gain)

            # Compute the frequency response of the filter

            w, h = freqz(hbp\_pm, worN=8000, fs=Fs)

            fig, axs = plt.subplots(1, 3, figsize=(14, 4))

            # Plot the impulse response

            axs[0].plot(hbp\_pm, label='Bandpass Filter')

            axs[0].set\_title('Impulse Response')

            axs[0].set\_xlabel('Sample')

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

            axs[0].legend()

            axs[0].grid(True)

            # Plot the frequency response

            axs[1].plot(w, 20 \* np.log10(abs(h)))

            axs[1].set\_title('Frequency Response')

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

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

            axs[1].grid(True)

            # Convolve the signal with the filter coefficients

            s\_pm = np.convolve(s, hbp\_pm, 'same')

            # Plot the Power Spectral Density (PSD) of the filtered signal

            f, Pxx\_spec = welch(s\_pm, Fs, 'flattop', 1024, scaling='spectrum')

            axs[2].semilogy(f, np.sqrt(Pxx\_spec))

            axs[2].set\_xlabel('Frequency [Hz]')

            axs[2].set\_ylabel('Linear spectrum [V RMS]')

            axs[2].set\_title('Power Spectrum (Welch)')

            axs[2].grid(True)

            plt.tight\_layout()

            plt.show()

# Attach the update\_plots function to the frequency inputs

f1\_input.observe(update\_plots, names='value')

f2\_input.observe(update\_plots, names='value')

# Display widgets

input\_ui = widgets.VBox([f1\_input, f2\_input])

ui = widgets.HBox([input\_ui], layout=Layout(align\_items='center'))

out = widgets.VBox([ui, output\_area])

display(out)

# Call update\_plots initially

update\_plots(None)

**Επιστροφή στην υλοποίηση**

#### Εφαρμογή Φίλτρου με 2 passbands:

Σχεδιασμός και υλοποίηση φίλτρου με δύο passbands στο MATLAB, με απεικόνιση της φασματικής του απόκρισης.

**Κώδικας – Chapter 1 - Εφαρμογή Φίλτρου με 2 passbands**

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import firwin2, freqz

# Sampling frequency

Fs = 8192

# Define the target frequency response

# Pass frequencies between 0-500 Hz and 1100-3000 Hz

target\_freqs = [0, 400, 500, 1100, 1200, 2900, 3000, Fs/2]

target\_response = [1, 1, 0, 0, 1, 1, 0, 0]  # 1 in passbands, 0 in stopbands

# Filter order

numtaps = 350

# Design the FIR filter using firwin2

filter\_taps = firwin2(numtaps, target\_freqs, target\_response, fs=Fs)

# Calculate the frequency response of the filter

w, h = freqz(filter\_taps, worN=2000, fs=Fs)

# Plot the frequency response

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)

plt.axhline(-60, color='red', linestyle='--', label="60 dB Attenuation", linewidth=1)

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()

**Επιστροφή στην υλοποίηση**

### Chapter 3: Optimal digital detection – Matched filters

#### Δημιουργία τυχαίων σημάτων:

Δημιουργία τυχαίων σημάτων και παρουσίαση του ιστογράμματός τους.

**Κώδικας – Chapter 3 - Δημιουργία τυχαίων σημάτων**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display

import time

# Function to generate histogram based on k and d inputs

def generate\_histogram(k, d):

    M = 40000

    L = 2 \*\* k

    # Generate random data based on k and d

    x = (2 \* np.floor(L \* np.random.rand(M)) - L + 1) \* d / 2

    bins = np.arange(-L \* d / 2, L \* d / 2 + 2 \* d / 2, d)

    A = np.arange(-L \* d / 2 + d / 2, L \* d / 2, d)

    # Plot histogram

    fig, ax = plt.subplots(1, 1, figsize=(14, 4))

    ax.hist(x, bins=bins, edgecolor='white', color='#1F77B4')

    ax.set\_xticks(A)

    ax.set\_xlabel("Integers")

    ax.set\_ylabel("Frequency")

    ax.set\_title("Histogram of array x elements")

    plt.show()

# UI Components for input

k\_input = widgets.IntText(

    value=3,  # default value

    description='k:',

    continuous\_update=False

)

d\_input = widgets.FloatText(

    value=1,  # default value

    description='d:',

    continuous\_update=False

)

# Arrange inputs in a VBox

inputs = widgets.VBox([k\_input, d\_input])

# Interactive output to update the plot based on k and d values

out = widgets.interactive\_output(generate\_histogram, {'k': k\_input, 'd': d\_input})

# Display the UI and output

display(inputs, out)

**Επιστροφή στην υλοποίηση**

#### Εφαρμογή θορύβου:

Επίδραση του θορύβου στο σήμα με τη ρύθμιση του λόγου Eb/No μέσω ενός διαδραστικού slider και τον υπολογισμό του ιστόγραμματος για διάφορες τιμές.

**Κώδικας – Chapter 3 - Εφαρμογή θορύβου**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display

# Function to generate histogram based on k, Eb/N0, and d inputs

def generate\_histogram(k, EbN0\_db, d):

    M = 60000  # Number of symbols

    nsamp = 16  # Number of samples per symbol

    L = 2 \*\* k  # Number of levels

    SNR\_db = EbN0\_db - 10 \* np.log10(nsamp / (2 \* k))  # Calculate SNR in dB

    SNR = 10 \*\* (SNR\_db \* 0.1)  # Convert SNR from dB to linear scale

    # Generate random signal with L levels

    x = (2 \* np.floor(L \* np.random.rand(M)) - L + 1) \* d / 2

    # Power of the original signal

    Px = (d \*\* 2 / 4) \* (L \*\* 2 - 1) / 3

    Measured\_x = np.sum(x \*\* 2) / len(x)

    # Upsample the signal

    y = np.repeat(x, nsamp)

    # Generate Gaussian noise

    noise = np.random.normal(0, np.sqrt(Measured\_x / SNR), len(y))

    y\_noisy = y + noise

    # Matched filter (average over nsamp samples)

    y\_reshaped = np.reshape(y\_noisy, (M, nsamp))

    matched = np.ones((nsamp, 1))

    z = np.matmul(y\_reshaped, matched) / nsamp

    # Set up the bins for the histogram

    A = np.arange(-(L - 1) \* d / 2, L \* d / 2, d)

    # Plot the histogram

    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([f"Eb/N0 = {EbN0\_db} dB"])

    ax.set\_title('Histogram of the Noisy Signal')

    plt.show()

# UI Components for input

k\_input = widgets.IntText(value=3, description='k:', continuous\_update=False)

d\_input = widgets.IntText(value=5, description='d:', continuous\_update=False)

EbN0\_db\_slider = widgets.FloatSlider(value=12, min=0, max=20, step=0.1, description='Eb/N0 (dB):', continuous\_update=False)

# Arrange inputs in a VBox

inputs = widgets.VBox([k\_input, d\_input, EbN0\_db\_slider])

# Interactive output to update the plot based on k, d, and Eb/N0 values

out = widgets.interactive\_output(generate\_histogram, {'k': k\_input, 'd': d\_input, 'EbN0\_db': EbN0\_db\_slider})

# Display the UI and output

display(inputs, out)

**Επιστροφή στην υλοποίηση**

#### ASK Bit Error Rate (BER) Visualization:

Η διαδραστική δυνατότητα σύγκρισης διαμορφώσεων Μ-ASK με επιλογές για πειραματική και θεωρητική τιμή του BER. Να μπορούν οι χρήστες να επιλέξουν μεταξύ διαφορετικών διαμορφώσεων, δειγμάτων ανά σύμβολο, και τύπου matched filter για να δουν τα αποτελέσματα σε πραγματικό χρόνο.

**Κώδικας – Chapter 3 - ASK Bit Error Rate (BER) Visualization**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets, Output, RadioButtons, Checkbox, Dropdown

from IPython.display import display

from scipy.special import erfc

from scipy.signal import upfirdn

# Function to calculate the number of errors in ASK modulation

def ask\_errors\_sin(k, M, nsamp, EbN0\_db, matched\_filter\_type='Normal'):

    # Determine number of levels in ASK constellation

    L = 2\*\*k

    # Calculate signal-to-noise ratio (SNR) in dB and linear scale

    SNR\_db = EbN0\_db - 10 \* np.log10(nsamp / (2 \* k))

    SNR\_linear = 10 \*\* (SNR\_db / 10)

    # Generate random ASK symbols and apply cosine pulse shaping

    x = 2 \* np.floor(L \* np.random.rand(M)) - L + 1

    h = np.cos(2 \* np.pi \* np.arange(1, nsamp + 1) / nsamp)

    h = h / np.sqrt(np.sum(h\*\*2))

    # Apply pulse shaping filter using upsample-filter-downsample method

    y = upfirdn(h, x, up=nsamp)

    y = y[:M \* nsamp]

    # Calculate noise variance and generate noise

    P\_x = np.mean(y \*\* 2)

    noise\_variance = P\_x / SNR\_linear

    noise = np.random.normal(0, np.sqrt(noise\_variance), len(y))

    # Add noise to signal

    y\_noisy = y + noise

    # Apply matched filter (normal or reversed) and sample at decision points

    matched = h[::-1] if matched\_filter\_type == 'Reversed' else h

    yrx = np.convolve(y\_noisy, matched, mode='full')

    z = yrx[nsamp - 1:M \* nsamp:nsamp]

    # Decision based on closest constellation levels

    levels = np.arange(-L + 1, L, 2)

    z\_decided = levels[np.abs(levels[:, None] - z).argmin(axis=0)]

    # Return the number of symbol errors

    return np.count\_nonzero(x != z\_decided)

# Set experiment parameters

M = 10000

EbN0\_db = np.arange(0, 21, 2)

# Create interactive UI components

checkbox\_4qam = Checkbox(value=True, description='4-ASK')

checkbox\_8qam = Checkbox(value=False, description='8-ASK')

checkbox\_16qam = Checkbox(value=False, description='16-ASK')

nsamp\_dropdown = Dropdown(options=[4, 8, 16, 32, 64], value=16, description='Samples per Symbol:')

matched\_radio = RadioButtons(options=['Normal', 'Reversed'], value='Normal', description='Matched Filter:')

plot\_output = Output()

# Plot selected modulations based on user input

def plot\_selected\_modulations(change):

    with plot\_output:

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

        nsamp = nsamp\_dropdown.value

        matched\_filter\_type = matched\_radio.value

        plt.figure(figsize=(10, 7))

        colors = {'4-ASK': 'red', '8-ASK': 'green', '16-ASK': 'blue'}

        for k, checkbox, color in zip([2, 3, 4], [checkbox\_4qam, checkbox\_8qam, checkbox\_16qam], colors.values()):

            if checkbox.value:

                L = 2\*\*k

                modulation\_name = f'{L}-ASK'

                ber = np.zeros(len(EbN0\_db))

                for index, eb\_n0 in enumerate(EbN0\_db):

                    ber[index] = ask\_errors\_sin(k, M, nsamp, eb\_n0, matched\_filter\_type) / (M \* np.log2(L))

                plt.semilogy(EbN0\_db, ber, 'o', label=f'Experimental {modulation\_name}', color=color)

                ber\_theoretical = (((L-1)/L) \* erfc(np.sqrt(10\*\*(EbN0\_db / 10) \* (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')

        plt.xlabel("Eb/N0 (dB)")

        plt.ylabel("BER")

        plt.title(f'Theoretical and Experimental BER of ASK Modulations [nsamp={nsamp}]')

        plt.legend()

        plt.show()

# Attach the plot function to UI interactions

checkbox\_4qam.observe(plot\_selected\_modulations, names='value')

checkbox\_8qam.observe(plot\_selected\_modulations, names='value')

checkbox\_16qam.observe(plot\_selected\_modulations, names='value')

nsamp\_dropdown.observe(plot\_selected\_modulations, names='value')

matched\_radio.observe(plot\_selected\_modulations, names='value')

# Set up the UI layout

inputs1 = widgets.HBox([nsamp\_dropdown])

inputs2 = widgets.HBox([matched\_radio], layout=Layout(margin="0 0 0 20px"))

inputs12 = widgets.HBox([inputs1, inputs2], layout=Layout(align\_items='center'))

inputs3 = widgets.VBox([checkbox\_4qam, checkbox\_8qam, checkbox\_16qam])

inputs = widgets.VBox([inputs12, inputs3])

ui = widgets.HBox([inputs], layout=Layout(align\_items='center'))

# Display UI and initial plot

display(ui, plot\_output)

plot\_selected\_modulations(None)

**Επιστροφή στην υλοποίηση**

#### Ανάλυση με φίλτρα:

Οπτικοποιήσεις απόκρισης φίλτρων πάνω σε ορθογώνιο και ημιτονοειδές σήμα.

**Κώδικας – Chapter 3 - Ανάλυση με φίλτρα**

# Necessary imports for numerical calculations, plotting, and interactive UI

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets

from IPython.display import display

# Output widget to display the plot

output = widgets.Output()

# Function to plot filters based on nsamp input

def plot\_filters(change):

    # Get the current value of nsamp from the dropdown

    nsamp = nsamp\_input.value

    # Clear the previous output before plotting the new data

    with output:

        output.clear\_output(wait=True)

        # Generate orthogonal and sinusoidal signals, both normalized

        orthogonal = np.ones(nsamp) / np.sqrt(nsamp)

        sinusoidal = np.cos(2 \* np.pi \* np.arange(1, nsamp + 1) / nsamp)

        sinusoidal = sinusoidal / np.sqrt(np.sum(sinusoidal\*\*2))

        # Create matched filters by reversing the signal arrays

        matched\_orthogonal = orthogonal[::-1]

        matched\_sinusoidal = sinusoidal[::-1]

        # Create a 2x2 subplot layout for plotting the filters

        fig, axs = plt.subplots(2, 2, figsize=(14, 8))

        # Plot orthogonal signal

        markerline, stemlines, baseline = axs[0, 0].stem(orthogonal)

        plt.setp(stemlines, 'linewidth', 0.5)

        plt.setp(markerline, 'markersize', 4)

        axs[0, 0].set\_title('Orthogonal')

        axs[0, 0].set\_xlabel('Index')

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

        # Plot sinusoidal signal

        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')

        axs[0, 1].set\_xlabel('Index')

        axs[0, 1].set\_ylabel('Amplitude')

        # Plot 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')

        axs[1, 0].set\_xlabel('Index')

        axs[1, 0].set\_ylabel('Amplitude')

        # Plot 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')

        axs[1, 1].set\_xlabel('Index')

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

        # Adjust layout to prevent overlap

        plt.tight\_layout()

        plt.show()

# Create a dropdown widget for selecting nsamp values

nsamp\_input = widgets.Dropdown(

    options=[8, 16, 32, 64],  # Available options for nsamp

    value=8,  # Default value

    description='nsamp:',

)

# Trigger the plot function when nsamp is changed

nsamp\_input.observe(plot\_filters, names='value')

# Create a vertical layout for the dropdown and output

inputs = widgets.VBox([nsamp\_input])

# Combine inputs and output into a horizontal layout

ui = widgets.HBox([inputs], layout=Layout(align\_items='center'))

out = widgets.VBox([ui, output])

# Display the UI

display(out)

# Initial plot when the page loads

plot\_filters(None)

**Επιστροφή στην υλοποίηση**

### Chapter 4: Spectral Characteristics of Digital Waveforms & Nyquist signaling

#### Ρυθμιζόμενες παράμετροι:

Εισαγωγή διαδραστικού τρόπου ρύθμισης παραμέτρων μήκους bitstream, roll-off factor, nsamp (δείγματα ανά σύμβολο), group delay, και σειρά του φίλτρου.

**Κώδικας – Chapter 4 - Ρυθμιζόμενες παράμετροι**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets, IntSlider, FloatSlider, Dropdown, IntText, Output

import scipy.signal

from math import log2

import random

# 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

# Output widget for dynamic updates

output = Output()

# Global variables for filter and parameters

global filt, g\_delay, g\_nsamp

# Function for interactive signal processing

def interactive\_signal\_processing(n\_bits, L, roll\_off, nsamp, delay):

    # Set global variables for filter and nsamp/delay values

    global filt, g\_delay, g\_nsamp

    g\_delay = delay

    g\_nsamp = nsamp

    # Generate random bitstream

    bitstream = generateRandomBits(n\_bits)

    # Determine ASK amplitude and number of bits per symbol

    A = L - 1

    k = int(log2(L))

    # Generate signal levels and symbols using Gray encoding

    y\_levels = createLevels(A, L)

    symbols = createSymbols(k, bitstream)

    gray\_encoding = grayCode(k)

    # Convert symbols to corresponding Gray code levels

    x\_gray = [y\_levels[gray\_encoding.index(symbol)] for symbol in symbols]

    # Generate root-raised cosine filter and apply it to the signal

    t\_filter, filt = rootRaisedCosine(nsamp, roll\_off, delay)

    y = upSample(x\_gray, nsamp)

    y\_transmitted = scipy.signal.convolve(y, filt)

    y\_received = scipy.signal.convolve(y\_transmitted, filt)

    # Downsample received signal and remove delay effects

    y\_final = downSample(y\_received, nsamp)

    y\_final = y\_final[2 \* delay: len(y\_final) - 2 \* delay]

    # Plot the results

    with output:

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

        fig = plt.figure(figsize=(20, 12))  # Create a figure object

        # First subplot - Root Raised Cosine Filter

        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 - Continuous signal with stems

        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')

        # 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 for stems

        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 - Signal Visualization

        ax3 = fig.add\_subplot(2, 2, 3)

        t = np.arange(0, len(y[:10]))

        ax3.plot(t, y\_final[:10])

        ax3.stem(t, x\_gray[:10])

        ax3.set\_title('Signal Visualization')

        ax3.legend(['Received','Transmitted'])

        ax3.set\_xlabel('Time')

        ax3.set\_ylabel('Amplitude')

        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)

        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 layout

        plt.show()

# Widget setup for interactive sliders and dropdowns

n\_bits\_slider = IntSlider(min=10000, max=100000, step=10000, value=10000, description='Bitstream Length', layout=Layout(width='100%'), continuous\_update=False)

roll\_off\_slider = FloatSlider(min=0.1, max=1.0, step=0.1, value=0.4, description='Roll-off Factor', layout=Layout(width='100%'), continuous\_update=False)

L\_dropdown = Dropdown(options=[2\*\*i for i in range(1, 6)], value=2, description='ASK Levels', continuous\_update=False)

nsamp\_slider = IntSlider(min=10, max=40, step=1, value=20, description='nsamp', layout=Layout(width='100%'), continuous\_update=False)

delay\_slider = IntSlider(min=1, max=10, step=1, value=5, description='Group Delay', layout=Layout(width='100%'), continuous\_update=False)

# Display the filter order based on nsamp and delay

filter\_order\_display = IntText(value=2 \* nsamp\_slider.value \* delay\_slider.value, description='Filter Order:', disabled=True)

# Grouping inputs into a VBox

inputs = widgets.VBox([n\_bits\_slider, roll\_off\_slider, nsamp\_slider, delay\_slider, filter\_order\_display, L\_dropdown])

# Function to update filter order based on nsamp and delay values

def update\_filter\_order(\*args):

    filter\_order = 2 \* nsamp\_slider.value \* delay\_slider.value

    filter\_order\_display.value = filter\_order

# Observers to update filter order when nsamp or delay changes

nsamp\_slider.observe(update\_filter\_order, 'value')

delay\_slider.observe(update\_filter\_order, 'value')

# Call interactive signal processing function when parameters change

def on\_change(\*args):

    interactive\_signal\_processing(n\_bits\_slider.value, L\_dropdown.value, roll\_off\_slider.value, nsamp\_slider.value, delay\_slider.value)

# Observe changes in the widgets and trigger signal processing

n\_bits\_slider.observe(on\_change, 'value')

roll\_off\_slider.observe(on\_change, 'value')

L\_dropdown.observe(on\_change, 'value')

nsamp\_slider.observe(on\_change, 'value')

delay\_slider.observe(on\_change, 'value')

# Display the UI and output area

display(inputs, output)

# Run the initial signal processing with default values

interactive\_signal\_processing(n\_bits\_slider.value, L\_dropdown.value, roll\_off\_slider.value, nsamp\_slider.value, delay\_slider.value)

**Επιστροφή στην υλοποίηση**

#### BER καμπύλες:

Με δυνατότητα επιλογής επιπέδου M-ASK, εμφάνισης των θεωρητικών και πειραματικών καμπυλών σε σχέση με το Eb/N0 και σύγκριση μεταξύ κωδικοποίησης Gray και Natural.

**Κώδικας – Chapter 4 - BER καμπύλες**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, VBox, HBox, IntSlider, FloatSlider, Dropdown, Output, interactive

from IPython.display import display, clear\_output

import scipy.signal

from math import log2, erfc, sqrt

import random

# 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

def interactive\_signal\_processing\_with\_natural(n\_bits=80000, roll\_off=0.7, nsamp=16, delay=4, L=16):

    # 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)

    plt.show()

# Creating sliders and dropdown

n\_bits\_slider = 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\_slider = 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\_slider = IntSlider(min=8, max=32, step=1, value=16, description='nsamp', style={'description\_width': 'initial'}, layout=Layout(width='100%'), continuous\_update=False)

delay\_slider = IntSlider(min=1, max=8, step=1, value=4, description='Group Delay', style={'description\_width': 'initial'}, layout=Layout(width='100%'), continuous\_update=False)

L\_dropdown = Dropdown(options=[2\*\*i for i in range(1, 6)], value=16, description='ASK Levels', style={'description\_width': 'initial'}, continuous\_update=False)

# Reordering the widgets

interactive\_plot\_with\_natural = interactive(interactive\_signal\_processing\_with\_natural, n\_bits=n\_bits\_slider, roll\_off=roll\_off\_slider, nsamp=nsamp\_slider, delay=delay\_slider, L=L\_dropdown)

input\_widgets = VBox([n\_bits\_slider, roll\_off\_slider, nsamp\_slider, delay\_slider, L\_dropdown], layout=Layout(flex='1 1 auto', width='auto'))

plot\_output = interactive\_plot\_with\_natural.children[-1]  # The output plot

# Create a VBox that includes both the input widgets and the loading animation

inputs\_and\_loader = HBox([input\_widgets])

# Create an HBox to hold everything in a horizontal layout

ui = VBox([inputs\_and\_loader, plot\_output])

# Display the UI components

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

display(ui)

**Επιστροφή στην υλοποίηση**

### Chapter 5: Digital Modulation QAM and PSK

#### QAM και PSK Constellations:

Διαδραστικά διαγράμματα αστερισμών για L-QAM και M-PSK. Οι χρήστες να μπορούν να επιλέξουν το επίπεδο διαμόρφωσης και να δουν τις αντίστοιχες θέσεις των συμβόλων στον αστερισμό.

**Κώδικας – Chapter 5 - QAM Constellation**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets, IntSlider, FloatSlider, Dropdown, VBox, HBox, interactive, Output, interact

from IPython.display import display, clear\_output

import time

# Function to generate QAM constellation points

def generate\_qam\_constellation(L):

    # Calculate total constellation points and bit length per symbol

    M = L \* L

    l = int(np.log2(L))

    # Generate x and y values for the grid of points in the QAM constellation

    x = np.arange(-(L - 1), L, 2)

    y = np.arange(-(L - 1), L, 2)

    # Create a grid of constellation points

    xv, yv = np.meshgrid(x, y)

    mapping = xv + 1j \* yv  # Combine in-phase and quadrature components

    mapping = mapping.flatten()  # Flatten the grid to a 1D array

    return mapping  # Return the QAM constellation points

# Function to plot the QAM constellation

def plot\_qam\_constellation(L):

    # Generate QAM points based on L value

    mapping = generate\_qam\_constellation(L)

    M = L \* L

    l = int(np.log2(L))

    # Plotting the QAM constellation

    plt.figure(figsize=(10, 7))

    plt.scatter(mapping.real, mapping.imag)  # Plot in-phase vs quadrature components

    # Add binary labels for points if L is small enough

    if L < 16:

        # Generate Gray code labels for the points

        labels = [bin(i)[2:].zfill(2 \* l) for i in range(M)]

        dx, dy = -0.5, 0.3  # Label offsets to avoid overlap

        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))

    # Set plot limits and labels

    plt.grid(True)

    plt.xlim(-1 \* L, 1 \* L)

    plt.ylim(-1 \* L, 1 \* L)

    plt.title(f"{L}x{L} QAM Constellation")

    plt.xlabel("In-phase")

    plt.ylabel("Quadrature")

    plt.show()

# Dropdown widget for selecting L value (size of the QAM constellation)

L\_dropdown = Dropdown(options=[2, 4, 8, 16, 32, 64], value=8, description='L-QAM:', continuous\_update=False)

# Create interactive widget to update plot based on dropdown selection

interactive\_plot = interact(plot\_qam\_constellation, L=L\_dropdown)

**Επιστροφή στην υλοποίηση**

import numpy as np

**Κώδικας – Chapter 5 - PSK Constellation**

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets, IntSlider, FloatSlider, Dropdown, VBox, HBox, interactive, Output, interact

from IPython.display import display, clear\_output

import time

# Function to generate PSK constellation points for a given M

def generate\_psk\_constellation(M):

    # Calculate the number of bits per symbol

    k = int(np.log2(M))

    # Initialize starting phase angles (for 4-PSK)

    ph1 = np.pi / 4

    theta = np.array([ph1, -ph1, np.pi - ph1, -np.pi + ph1])

    # Generate the initial 4-PSK constellation points

    mapping = np.exp(1j \* theta)

    # Expand the constellation for higher-order PSK (M > 4)

    if k > 2:

        for j in range(3, k + 1):

            # Halve the angle for the next level of PSK

            theta = theta / 2

            # Generate new PSK points

            mapping = np.exp(1j \* theta)

            # Mirror the points to fill the other half of the constellation

            mapping = np.concatenate((mapping, -np.conjugate(mapping)))

            # Update theta for the next iteration

            theta = np.real(np.log(mapping) / 1j)

    # Return the PSK constellation points

    return mapping

# Function to plot the PSK constellation

def plot\_psk\_constellation(M):

    # Generate PSK points based on M value

    constellation = generate\_psk\_constellation(M)

    # Calculate the number of bits per symbol

    k = int(np.log2(M))

    # Plotting the PSK constellation

    plt.figure(figsize=(10, 7))

    plt.scatter(np.real(constellation), np.imag(constellation))  # Plot the real vs imaginary components

    plt.grid(True)

    # Add title and axis labels

    plt.title(f'{M}-PSK Constellation')

    plt.xlabel('In-Phase')

    plt.ylabel('Quadrature')

    # Add grid lines through the origin

    plt.axhline(0, color='gray', linewidth=0.5)

    plt.axvline(0, color='gray', linewidth=0.5)

    # 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)),  # Binary label

                 bbox=dict(facecolor='red', alpha=0.5))

    # Display the plot

    plt.show()

# Dropdown widget for selecting M value (order of PSK)

M\_dropdown = Dropdown(

    options=[4, 8, 16, 32, 64],  # Available PSK options

    value=16,  # Default value

    description='M-PSK:'  # Dropdown label

)

# Create an interactive widget to update the PSK plot based on the dropdown selection

interactive\_plot\_psk = interact(plot\_psk\_constellation, M=M\_dropdown)

**Επιστροφή στην υλοποίηση**

#### QAM και PSK Καμπύλες BER:

Δημιουργία γραφημάτων για τις καμπύλες BER τόσο για QAM όσο και για PSK διαμορφώσεις, όπου οι χρήστες μπορούν να συγκρίνουν την θεωρητική και πειραματική καμπύλη και να ρυθμίσουν παραμέτρους όπως το roll-off, τις συχνότητες f1 και f2, και το bitrate για την ανάλυση της πιθανότητας σφάλματος.

**Κώδικας – Chapter 5 - QAM BER**

import numpy as np

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets, Dropdown, VBox, HBox, FloatText, interactive

import scipy.signal

import time

import scipy.special

# Function to generate the Root Raised Cosine (RRC) filter

def rootRaisedCosine(nsamp, roll\_off, delay):

    # Fundamental frequency and bandwidth settings

    F0 = 0.5 / nsamp  # Base frequency relative to nsamp

    Br = 1  # Bandwidth rate

    Fs = Br \* nsamp  # Sampling frequency

    Td = 1 / Br  # Symbol time

    Ts = 1 / Fs  # Sampling period

    # Frequency boundaries based on roll-off factor

    F1 = F0 \* (1 - roll\_off)

    F2 = F0 \* (1 + roll\_off)

    # Calculate filter order based on nsamp and delay

    filter\_order = 2 \* nsamp \* delay

    # Generate time vector for the filter

    t = np.arange(0, filter\_order, Td)

    h = []  # Initialize filter coefficients list

    # Loop to calculate filter coefficients

    for i in range(len(t)):

        # Shift time so it is centered around zero

        t\_shifted = t[i] - filter\_order / 2

        # Handle the case when time is zero

        if t\_shifted == 0:

            h.append(np.sqrt(2 \* F0) \* (1 + roll\_off \* ((4 / np.pi) - 1)))

        # Handle special cases related to roll-off

        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)))

        # General case for other time values

        else:

            # Calculate the filter coefficient based on the time-shifted value

            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)

            # Append the calculated filter coefficient

            h.append(factor1 \* (factor2 + factor3))

    # Return the filter coefficients (h)

    return h

# Function to calculate Bit Error Rate (BER) for QAM modulation

def ber\_qam(EbNo, M, roll\_off, F1, F2, Br):

    # Convert frequencies from MHz to Hz and bandwidth from Mbps to Hz

    F1 = F1 \* 1e6

    F2 = F2 \* 1e6

    Br = Br \* 1e6

    # Calculate bandwidth and carrier frequency

    W = F2 - F1  # Bandwidth in Hz

    fc = F1 + W / 2  # Carrier frequency

    # Calculate the number of samples per symbol

    nsamp = int(np.ceil(2 \* F2 / Br)) + 7

    # Determine QAM modulation parameters

    L = int(np.sqrt(M))  # Size of constellation

    l = np.log2(L)  # Number of bits per symbol in one dimension

    k = 2 \* l  # Total bits per QAM symbol (I and Q)

    Nsymb = 10000  # Number of symbols to simulate

    # Calculate Signal-to-Noise Ratio (SNR) in dB

    SNR = EbNo - 10 \* np.log10(nsamp / k / 2)

    # Generate QAM constellation points

    core = [1 + 1j, 1 - 1j, -1 + 1j, -1 - 1j]

    mapping = core[:]

    # Extend the constellation for higher-order QAM

    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 bit sequence

    x = np.floor(2 \* np.random.rand(int(k \* Nsymb), 1))

    x\_temp = np.reshape(x, (int(len(x) / k), int(k)))

    # Map bits to QAM symbols

    xsym = []

    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.append(a)

    # Map symbols to constellation points

    y = []

    for n in range(len(xsym)):

        y.append(mapping[xsym[n]])

    # Set filter parameters

    delay = 10

    filtorder = delay \* nsamp \* 2

    # Apply root raised cosine filter to the signal

    shaping\_filter = rootRaisedCosine(nsamp, roll\_off, delay)

    ytx = scipy.signal.upfirdn([1], y, nsamp)  # Upsample signal

    ytx = np.convolve(ytx, shaping\_filter)

    # Modulate the signal with the carrier frequency

    m = np.arange(1, len(ytx) + 1)

    s = np.real(np.multiply(ytx, np.exp(1j \* 2 \* np.pi \* fc \* m / nsamp)))

    # Calculate transmitted signal power

    s\_matrix = np.matrix(s)

    s\_matrix = s\_matrix.getH()

    s\_list = s\_matrix.tolist()

    Ps = 10 \* np.log10(np.matmul(s, s\_list) / len(s))

    # Calculate noise power based on SNR

    Pn = Ps - SNR

    n = np.sqrt(10\*\*(Pn / 10)) \* np.random.randn(1, len(ytx))

    # Add noise to the signal

    snoisy = s + n

    # Demodulate the received signal

    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]

    yrx = yrx[2 \* delay: len(yrx) - 2 \* delay]

    # Separate real and imaginary parts (I and Q components)

    yi = np.real(yrx)

    yq = np.imag(yrx)

    # Decision making: Map received values back to the closest constellation points

    q = np.arange(-L + 1, L, 2)

    for n in range(len(yrx)):

        yi[n] = q[np.argmin(np.abs(q - yi[n]))]

        yq[n] = q[np.argmin(np.abs(q - yq[n]))]

    # Calculate Bit Error Rate (BER)

    error = 0

    for i in range(len(yrx)):

        if y[i] != yi[i] + 1j \* yq[i]:

            error += 1

    # Return the BER

    return error / len(x)

# Function to plot BER curve for QAM modulation

def plot\_ber\_qam(M, roll\_off, F1, F2, Br):

    # Check if the frequency range is valid

    if F1 >= F2:

        print("Warning: F1 should be less than F2.")

        return

    # Initialize lists to store experimental and theoretical BER values

    ber\_exp = []

    ber\_th = []

    # Calculate the number of levels for the QAM constellation

    L = int(np.sqrt(M))

    # Loop through Eb/N0 values (in dB)

    for i in range(1, 15):

        # Append the experimental BER calculated using the `ber\_qam` function

        ber\_exp.append(ber\_qam(i, M, roll\_off, F1, F2, Br))

        # Calculate and append the theoretical BER for QAM

        ber\_th.append(((L - 1) / (L \* np.log2(L)) \*

                       scipy.special.erfc(np.sqrt(3 \* np.log2(L) / (L \* L - 1) \* 10\*\*(i/10)))))

    # Plot the results

    plt.figure(figsize=(10, 8))

    # Plot theoretical BER as a line

    plt.semilogy(range(1, 15), ber\_th)

    # Plot experimental BER as points

    plt.semilogy(range(1, 15), ber\_exp, 'o')

    # Add labels, title, legend, and grid

    plt.legend(['Theoretical', 'Simulation'])

    plt.xlabel('Eb/N0 (dB)')  # X-axis label

    plt.ylabel('Bit Error Probability')  # Y-axis label

    plt.title(f'BER curve for {M}-QAM')  # Plot title

    plt.grid(which='both')  # Enable grid for both major and minor ticks

    plt.show()  # Display the plot

# 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)')

# Create interactive widget

interactive\_plot = interactive(plot\_ber\_qam, M=qam\_selector, roll\_off=roll\_off\_input, F1=F1\_input, F2=F2\_input, Br=Br\_input)

input\_widgets = VBox([qam\_selector, roll\_off\_input, F1\_input, F2\_input, Br\_input], layout=Layout(width='auto'))

plot\_output = interactive\_plot.children[-1]

# Display the UI components

display(input\_widgets, plot\_output)

**Επιστροφή στην υλοποίηση**

import numpy as np

**Κώδικας – Chapter 5 - PSK BER**

import matplotlib.pyplot as plt

from ipywidgets import Layout, widgets, Dropdown, VBox, HBox, interactive

from IPython.display import display, clear\_output

import scipy.signal

import scipy.special

import time

from scipy.signal import upfirdn

# Root Raised Cosine (RRC) filter function

def rootRaisedCosine1(nsamp, roll\_off, delay):

    # Time vector for filter based on delay and samples per symbol

    t = np.arange(-delay, delay + 1 / nsamp, 1 / nsamp)

    h = np.zeros(len(t))  # Initialize filter coefficients

    # Loop to calculate filter coefficients

    for i in range(len(t)):

        if t[i] == 0.0:  # Special case for t = 0

            h[i] = 1.0 - roll\_off + 4 \* roll\_off / np.pi

        elif roll\_off != 0 and t[i] == 1 / (4 \* roll\_off):  # Special case for specific t

            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):  # Symmetric case for t

            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:  # General case

            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  # Return filter coefficients

# Function to compute theoretical BER for M-PSK

def compute\_ber\_psk(EbNo\_dB, M1):

    EbNo\_linear = 10\*\*(EbNo\_dB / 10)  # Convert dB to linear scale

    if M1 == 2:  # BPSK case

        return 0.5 \* scipy.special.erfc(np.sqrt(EbNo\_linear))

    else:  # M-PSK case

        k = np.log2(M1)  # Bits per symbol

        return (1 / 4 \* k) \* scipy.special.erfc(np.sqrt(EbNo\_linear \* k) \* np.sin(np.pi / M1))

# Function to simulate BER for M-PSK

def ber\_psk\_simulation(EbNo\_dB, M1):

    Nsymb = 30000  # Number of symbols

    nsamp = 16  # Samples per symbol

    fc = 4  # Carrier frequency

    rolloff = 0.25  # Roll-off factor

    delay = 10  # Filter delay

    # Calculate SNR in dB

    SNR\_dB = EbNo\_dB - 10 \* np.log10(nsamp / np.log2(M1))

    # Generate RRC filter

    shaping\_filter = rootRaisedCosine1(nsamp, rolloff, delay)

    # Generate random symbols

    bits1 = np.random.randint(0, M1, Nsymb)

    # Map bits to PSK symbols

    symbols = np.exp(1j \* (2 \* np.pi \* bits1 / M1))

    # Upsample and filter transmitted signal

    ytx1 = upfirdn([1], symbols, nsamp)

    ytx1 = np.convolve(ytx1, shaping\_filter, mode='same')

    # Modulate with carrier frequency

    m1 = np.arange(len(ytx1))

    s1 = np.real(ytx1 \* np.exp(1j \* 2 \* np.pi \* fc \* m1 / nsamp))

    # Calculate signal power and noise power

    Ps = np.mean(np.abs(s1)\*\*2)

    SNR\_linear = 10\*\*(SNR\_dB / 10)

    Pn = Ps / SNR\_linear

    noise = np.sqrt(Pn / 4) \* (np.random.randn(len(s1)) + 1j \* np.random.randn(len(s1)))

    # Add noise to the signal

    snoisy = s1 + noise

    # Demodulate received signal

    yrx1 = snoisy \* np.exp(-1j \* 2 \* np.pi \* fc \* m1 / nsamp)

    yrx1 = np.convolve(yrx1, shaping\_filter, mode='same')

    yrx1 = yrx1[::nsamp]

    # Detect symbols by mapping angles back to symbol indices

    detected\_bits1 = np.angle(yrx1) \* M1 / (2 \* np.pi)

    detected\_bits1 = np.round(detected\_bits1) % M1

    # Calculate BER by comparing transmitted and received symbols

    bit\_errors1 = np.sum(bits1 != detected\_bits1)

    ber1 = bit\_errors1 / len(bits1)

    return ber1  # Return the BER

# Function to plot BER curve for PSK modulation

def plot\_ber\_psk(M):

    ber\_exp = []  # List for experimental BER values

    ber\_th = []  # List for theoretical BER values

    # Loop through Eb/N0 values in dB

    for i in range(1, 18):

        ber\_exp.append(ber\_psk\_simulation(i, M))  # Simulated BER

        ber\_th.append(compute\_ber\_psk(i, M))  # Theoretical BER

    # Plot the results

    plt.figure(figsize=(10, 8))

    # Plot theoretical BER curve

    plt.semilogy(range(1, 18), ber\_th)

    # Plot experimental BER points

    plt.semilogy(range(1, 18), ber\_exp, 'o')

    # Add labels, title, and grid

    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()  # Display the plot

# Define PSK options

psk\_options = {'BPSK': 2, 'QPSK': 4, '8-PSK': 8}

psk\_selector = widgets.Dropdown(options=psk\_options, value=4, description='PSK Type:')

# Create interactive widget

interactive\_plot = interactive(plot\_ber\_psk, M=psk\_selector)

input\_widgets = VBox([psk\_selector], layout=Layout(width='auto'))

plot\_output = interactive\_plot.children[-1]  # The output plot

# Create a VBox that includes both the input widgets and the loading animation

inputs = HBox([input\_widgets], layout=Layout(align\_items='center'))

# Create an HBox to hold everything in a horizontal layout

ui = VBox([inputs, plot\_output])

# Display the UI components

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

display(ui)

**Επιστροφή στην υλοποίηση**

#### Διαδραστική Φασματική Πυκνότητα Ισχύος:

Δυνατότητα οπτικοποίησης της φασματικής πυκνότητας ισχύος για διάφορες διαμορφώσεις QAM και συχνοτήτων f1 και f2.

**Κώδικας – Chapter 5 - Διαδραστική Φασματική Πυκνότητα Ισχύος**

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import upfirdn, convolve, welch

import ipywidgets as widgets

from IPython.display import display, clear\_output

# Function to generate Root Trapezium filter (used for shaping the signal)

def rtrapezium(nsamp, rolloff, delay):

    T = 1  # Symbol time

    # Time vector for the filter

    t = np.arange(-delay\*T, (delay\*T) + 1/nsamp, 1/nsamp)

    rrc = np.zeros\_like(t)  # Initialize filter coefficients array

    for i, ti in enumerate(t):

        if ti == 0.0:  # Special case when time equals zero

            rrc[i] = 1.0 - rolloff + 4\*rolloff/np.pi

        elif abs(ti) == T / (4 \* rolloff):  # Special case when t is related to 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:  # General case for other time values

            rrc[i] = (np.sin(np.pi \* ti \* (1 - rolloff) / T) +

                      4 \* rolloff \* ti \* np.cos(np.pi \* ti \* (1 + rolloff) / T)) / (

                      np.pi \* ti \* (1 - (4 \* rolloff \* ti / T) \*\* 2))

    return rrc / np.sqrt(np.sum(rrc\*\*2))  # Normalize filter energy

# Function to run QAM simulation

def run\_simulation(f1, f2, qam\_type):

    with output:  # Output to specific widget area

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

        # Parameters

        k = int(np.log2(qam\_type))  # Number of bits per symbol

        M = 2\*\*k  # Modulation order

        Nsymb = 30000  # Number of symbols

        pulse\_type = 1  # 1 for rtrapezium shaping filter, 0 for rectangular pulse

        nsamp = 32  # Oversampling factor

        fc = (f1 + f2) / 2  # Carrier frequency

        bandwidth = f2 - f1  # Signal bandwidth

        rolloff = bandwidth / (2 \* fc)  # Rolloff factor based on bandwidth

        EbNo = 10  # Eb/No in dB

        SNR = EbNo - 10 \* np.log10(nsamp / k / 2)  # SNR per signal sample

        # Phase and mapping initialization for QAM

        ph1 = np.pi / 4

        theta = np.array([ph1, -ph1, np.pi - ph1, -np.pi + ph1])

        mapping = np.exp(1j \* theta)

        # Generate higher-order QAM mappings if necessary

        if k > 2:

            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: Generate random symbols and map to QAM constellation points

        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 binary to decimal

        y = mapping[xsym]  # Map to QAM symbols

        # Define shaping filter

        if pulse\_type == 1:  # Nyquist pulse -- rtrapezium

            delay = 8  # Group delay (# of symbol periods)

            shaping\_filter = rtrapezium(nsamp, rolloff, delay)

        else:  # Rectangular pulse shaping

            delay = 0.5

            shaping\_filter = np.ones(nsamp) / np.sqrt(nsamp)  # Normalized rectangular pulse

        # Transmitted signal: Upsample and apply shaping filter

        ytx = upfirdn([1], y, nsamp)

        ytx = convolve(ytx, shaping\_filter, mode='same')

        # Quadrature modulation with carrier frequency

        m = np.arange(len(ytx))

        s = np.real(ytx \* np.exp(1j \* 2 \* np.pi \* fc \* m / nsamp))

        # 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: Demodulation and filtering

        yrx = 2 \* snoisy \* np.exp(-1j \* 2 \* np.pi \* fc \* m / nsamp)

        yrx = convolve(yrx, shaping\_filter, mode='same')

        # Spectrum plot of the received signal

        f, Pxx\_den = welch(np.real(s), fs=nsamp, nperseg=1024)

        Pxx\_den = 10 \* np.log10(Pxx\_den)

        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 (V^2/Hz)')

        plt.xlim(0, max(f1, f2) + 5)

        plt.grid()

        plt.show()

# Widgets for f1, f2, and QAM type

f1\_widget = widgets.FloatText(value=6.75, description='f1:')  # Start frequency

f2\_widget = widgets.FloatText(value=9.25, description='f2:')  # End frequency

qam\_widget = widgets.Dropdown(options=[4, 16, 64], value=16, description='QAM Type:')  # QAM type dropdown

# Callback function when any widget value changes

def on\_value\_change(change):

    run\_simulation(f1\_widget.value, f2\_widget.value, qam\_widget.value)

# Observe changes in widget values

f1\_widget.observe(on\_value\_change, names='value')

f2\_widget.observe(on\_value\_change, names='value')

qam\_widget.observe(on\_value\_change, names='value')

# Output widget for displaying simulation results

output = widgets.Output()

# Display the widgets and output area

display(qam\_widget, f1\_widget, f2\_widget, output)

# Run initial simulation

run\_simulation(f1\_widget.value, f2\_widget.value, qam\_widget.value)

**Επιστροφή στην υλοποίηση**

#### Διαδραστικός Υπολογισμός Παραμέτρων:

Δυνατότητα υπολογισμού παραμέτρων, με βάση τις παραμέτρους (R, M και roll-off) που επιλέγει ο χρήστης, όπως ο ρυθμός.

**Κώδικας – Chapter 5 - Διαδραστικός Υπολογισμός Παραμέτρων**

import numpy as np

import time

import ipywidgets as widgets

from IPython.display import display, clear\_output

from ipywidgets import HBox, VBox, Layout

# Function to calculate maximum achievable transmission rate R' and percentage increase

def calculate\_R\_and\_percentage\_increase(M, a, R):

    W = 2.5 \* (10 \*\* 6)  # Fixed bandwidth in Hz

    log2M = np.log2(M)  # Calculate log2(M) for QAM modulation

    R\_prime = (log2M \* W) / (1 + a)  # Maximum achievable rate in bps

    R\_prime\_mbps = R\_prime / (10 \*\* 6)  # Convert rate to Mbps

    percentage\_increase = ((R\_prime\_mbps - R) / R) \* 100  # Calculate percentage increase

    return R\_prime\_mbps, percentage\_increase

# Create input widgets for user input

R\_input = widgets.FloatText(description='R (Mbps):', value=8.0)  # Input for current rate (R)

M\_input = widgets.FloatText(description='M:', value=16)  # Input for modulation order (M)

a\_input = widgets.FloatText(description='α\' (roll-off):', value=0.125)  # Input for roll-off factor (a)

# Output widget to display the calculated results

output\_vals = widgets.Output()

# Function to update the result when input values change

def update\_result(change=None):

    with output\_vals:

        output\_vals.clear\_output()  # Clear previous output

        R = R\_input.value  # Get current rate from input

        M = M\_input.value  # Get modulation order from input

        a = a\_input.value  # Get roll-off factor from input

        R\_prime\_mbps, percentage\_increase = calculate\_R\_and\_percentage\_increase(M, a, R)  # Calculate rate and increase

        # Display the results

        print(f"Maximum Achievable Rate (R') = {R\_prime\_mbps:.3f} Mbps")

        print(f"Percentage Increase = {percentage\_increase:.2f}%")

# Set observers to call update\_result whenever an input value changes

R\_input.observe(update\_result, names='value')

M\_input.observe(update\_result, names='value')

a\_input.observe(update\_result, names='value')

# Organize widgets into a vertical box layout

input\_widgets = VBox([R\_input, M\_input, a\_input], layout=Layout(width='auto'))

# Create a UI layout that includes input widgets and output

ui = VBox([input\_widgets, output\_vals])

# Display the UI components

clear\_output(wait=True)

display(ui)

# Perform the initial calculation

update\_result()

**Επιστροφή στην υλοποίηση**

### Chapter 6: Digital Modulation FSK and MSK

#### Αριθμός σφαλμάτων για Eb/No:

Γράφημα στο οποίο ο χρήστης ρυθμίζει ένα εύρος του Eb/No, και στη συνέχεια εμφανίζεται η καμπύλη που απεικονίζει τον αριθμό των σφαλμάτων σε σχέση με το Eb/No για συστήματα FSK.

**Κώδικας – Chapter 6 - Αριθμός σφαλμάτων για Eb/No**

import numpy as np

import matplotlib.pyplot as plt

import ipywidgets as widgets

from ipywidgets import HBox, VBox, Layout, IntRangeSlider, interactive

from IPython.display import display, clear\_output

from commpy.channels import awgn  # Import AWGN (Additive White Gaussian Noise) function from commpy

import time

# Parameters

bits\_per\_symbol = 4  # Number of bits per symbol for M-FSK

num\_symbols = 1000  # Number of symbols to simulate

num\_samples = 80  # Number of samples per symbol (oversampling factor)

# Derived parameters

M = 2 \*\* bits\_per\_symbol  # Number of different symbols (M-FSK)

baud\_rate = 1  # Baud rate (symbol rate)

carrier\_freq = 2 \* M \* baud\_rate  # Carrier frequency for FSK signal

# Calculate total number of bits and symbol/sampling periods

num\_bits = bits\_per\_symbol \* num\_symbols  # Total number of bits

symbol\_period = 1 / baud\_rate  # Symbol period

sampling\_period = symbol\_period / num\_samples  # Sampling period

# M frequencies spaced within the coherent distance (Baud Rate)

frequencies = carrier\_freq + (baud\_rate / 2) \* (np.arange(1, M + 1) - (M + 1) / 2)

# Find the maximum frequency

max\_frequency = np.max(frequencies)

# Recalculate the number of samples to ensure the sampling frequency (Fs) meets Nyquist criteria

sampling\_freq = 2 \* max\_frequency

num\_samples = int(np.ceil(sampling\_freq / baud\_rate)) + 10  # Adjust number of samples

# Recalculate the sampling period based on updated num\_samples

sampling\_period = symbol\_period / num\_samples

# Generate random input bits and reshape them into symbols

bits = np.random.randint(0, 2, num\_bits)

bit\_matrix = bits.reshape((num\_symbols, bits\_per\_symbol))

# Time vectors

time\_vector = np.arange(0, len(bit\_matrix) \* symbol\_period, symbol\_period)  # Time vector for symbols

oversampling\_time\_vector = np.arange(0, symbol\_period, sampling\_period)  # Time vector for oversampling

# Generate the FSK signal

fsk\_signal = []

amplitude = np.sqrt(2 / symbol\_period / num\_samples)  # Normalize amplitude

for symbol\_index in range(len(bit\_matrix)):

    # Convert binary symbol to frequency index

    freq\_symbol = frequencies[int(''.join(map(str, bit\_matrix[symbol\_index])), 2)]

    time\_segment = (symbol\_index \* symbol\_period) + oversampling\_time\_vector

    fsk\_signal.append(np.sin(2 \* np.pi \* freq\_symbol \* time\_segment))  # FSK modulation

fsk\_signal = np.concatenate(fsk\_signal)  # Concatenate the signal

# Function to calculate number of errors for a range of Eb/No values

def calculate\_errors(EbNo\_range):

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

    EbNo\_values = list(range(EbNo\_range[0], EbNo\_range[1] + 1))  # Range of Eb/No values

    errors\_list = []  # List to store the number of errors for each Eb/No

    for EbNo in EbNo\_values:

        # Calculate SNR based on Eb/No, bits per symbol, and oversampling

        SNR = EbNo + 10 \* np.log10(bits\_per\_symbol) - 10 \* np.log10(num\_samples / 2)  # SNR in dB

        # Add AWGN to the FSK signal

        noisy\_signal = awgn(fsk\_signal, SNR)

        # FSK receiver (demodulation and decoding)

        received\_symbols = []

        for symbol\_index in range(len(noisy\_signal) // num\_samples):

            time\_segment = (symbol\_index \* symbol\_period) + oversampling\_time\_vector

            signal\_segment = noisy\_signal[symbol\_index \* num\_samples:(symbol\_index + 1) \* num\_samples]

            # Correlate with each possible frequency to find the best match

            match\_scores = []

            for freq in frequencies:

                signal\_template = np.sin(2 \* np.pi \* freq \* time\_segment)

                match\_scores.append(np.sum(signal\_segment \* signal\_template))

            decoded\_symbol = np.argmax(match\_scores)  # Find the symbol with the highest match score

            received\_symbols.append([int(bit) for bit in bin(decoded\_symbol)[2:].zfill(bits\_per\_symbol)])  # Decode symbol

        received\_symbols = np.array(received\_symbols).reshape((num\_symbols, bits\_per\_symbol))

        # Count bit errors by comparing transmitted and received symbols

        errors = np.sum(bit\_matrix != received\_symbols)

        errors\_list.append(errors)

    # Plot the number of errors against 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()

# Create a slider widget for Eb/No range

EbNo\_slider = IntRangeSlider(

    value=[0, 20],  # Initial range

    min=0,

    max=20,

    step=1,

    description='EbNo (dB):',

    continuous\_update=False,  # Update only on release of slider

    layout=Layout(width='99%')  # Slider layout

)

# Create an interactive widget to run the simulation when the slider changes

interactive\_plot = interactive(calculate\_errors, EbNo\_range=EbNo\_slider)

# Combine the slider and plot into a VBox layout

input\_widgets = VBox([EbNo\_slider], layout=Layout(flex='1 1 auto', width='auto'))

plot\_output = interactive\_plot.children[-1]  # Get the plot output

# Create a VBox layout containing both input widgets and plot output

ui = VBox([input\_widgets, plot\_output])

# Display the UI components

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

display(ui)

**Επιστροφή στην υλοποίηση**

#### FSK Bit Error Rate:

Διαδραστική σύγκριση των καμπυλών BER για coherent και non coherent συστήματα FSK. Οι χρήστες μπορούν να δουν τις καμπύλες που αντιπροσωπεύουν την θεωρητική και πειραματική απόδοση.

**Κώδικας – Chapter 6 - FSK Bit Error Rate**

import numpy as np

import matplotlib.pyplot as plt

import ipywidgets as widgets

from ipywidgets import HBox, VBox, Layout, Dropdown, interactive

from IPython.display import display, clear\_output

from commpy.channels import awgn

from scipy.signal import convolve

import time

# Function to calculate bit errors in coherent FSK

def fsk\_errors(bps, Nsymb, ns, EbNo):

    # Input parameters

    M = 2 \*\* bps  # number of different symbols

    BR = 1  # Baud Rate

    fc = 2 \* M \* BR  # RF frequency

    # Derived parameters

    nb = bps \* Nsymb  # number of simulated data bits

    T = 1 / BR  # one symbol period

    Ts = T / ns  # oversampling period

    # M frequencies in "coherent" distance (BR)

    f = fc + (BR/2) \* (np.arange(1, M + 1) - (M + 1) / 2)

    # Calculate the maximum frequency

    fmax = np.max(f)

    # Recalculate ns to ensure Fs = ns \* BR > 2 \* fmax

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    # Recalculate the oversampling period

    Ts = T / ns

    # awgn channel

    SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)  # in dB

    # input data bits

    y = np.random.randint(0, 2, nb)

    x = y.reshape((Nsymb, bps))

    t = np.arange(0, len(x) \* T, T)  # time vector on the T grid

    tks = np.arange(0, T, Ts)  # oversampling time vector

    # FSK signal

    s = []

    A = np.sqrt(2 / T / ns)

    for k in range(len(x)):

        fk = f[int(''.join(map(str, x[k])), 2)]

        tk = (k \* T) + tks

        s.append(np.sin(2 \* np.pi \* fk \* tk))

    s = np.concatenate(s)

    # add noise to the FSK (passband) 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

# Non-coherent FSK error calculation

def fsk\_errors\_non\_coh(bps, Nsymb, ns, EbNo):

    M = 2 \*\* bps  # number of different symbols

    BR = 1  # Baud Rate

    fc = 2 \* M \* BR  # RF frequency

    # Derived parameters

    nb = bps \* Nsymb  # number of simulated data bits

    T = 1 / BR  # one symbol period

    Ts = T / ns  # oversampling period

    f = fc + BR \* (np.arange(1, M + 1) - (M + 1) / 2)  # M frequencies

    # Calculate the maximum frequency

    fmax = np.max(f)

    # Recalculate ns to ensure Fs = ns \* BR > 2 \* fmax

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    # Recalculate the oversampling period

    Ts = T / ns

    # AWGN channel

    SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)  # in dB

    # input data bits

    y = np.random.randint(0, 2, nb)  # Ensure integer dimensions

    x = y.reshape((-1, bps))

    t = np.arange(0, len(x) \* T, T)  # time vector on the T grid

    tks = np.arange(0, T, Ts)

    # FSK signal

    s = []

    A = np.sqrt(2 / T / ns)

    for k in range(len(x)):

        fk = f[int("".join(map(str, x[k])), 2)]

        tk = (k \* T) + tks

        s = np.concatenate((s, np.sin(2 \* np.pi \* fk \* tk)))

    # add noise to the FSK (passband) signal

    s = awgn(s, SNR)

    # FSK receiver

    # Non-coherent demodulation

    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[:len(sk)])

            smq = np.sum(sk \* sq[:len(sk)])

            sm.append(np.sqrt(smi \*\* 2 + smq \*\* 2))

        j = np.argmax(sm)

        xr = np.concatenate((xr, np.array(list(np.binary\_repr(j, width=bps)), dtype=int)))

    # count errors

    x\_reshaped = x.reshape(-1)

    xr\_reshaped = xr.reshape(-1)

    err = np.not\_equal(x\_reshaped, xr\_reshaped[:len(x\_reshaped)])

    errors = np.sum(err)

    return errors

# BER simulation function

def simulate\_ber(bps, Nsymb, ns, EbNo\_values, coherent=True):

    ber = []

    for EbNo in EbNo\_values:

        errors = 0

        for \_ in range(1):  # averaging over iterations

            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

# Theoretical BER for coherent FSK systems

def theoretical\_ber\_coh(EbNo, M):

    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")

# Theoretical BER for non-coherent FSK systems

def theoretical\_ber\_non\_coh(EbNo, M):

    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")

# Function to update the plot

def update\_plot(bps):

    Nsymb = 2000  # number of symbols

    ns = 100  # oversampling factor

    EbNo\_dB\_sim = np.arange(0, 10, 2)  # Eb/No values in dB for simulation

    EbNo\_dB\_theory = np.arange(0, 15, 1)  # Eb/No values in dB for theoretical

    EbNo\_sim = 10 \*\* (EbNo\_dB\_sim / 10)  # convert dB to linear for simulation

    EbNo\_theory = 10 \*\* (EbNo\_dB\_theory / 10)  # convert dB to linear for theoretical

    M = 2 \*\* bps  # M-ary FSK

    # Simulate BER

    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)

    # Theoretical BER

    ber\_coh\_theory = theoretical\_ber\_coh(EbNo\_theory, M)

    ber\_non\_coh\_theory = theoretical\_ber\_non\_coh(EbNo\_theory, M)

    # Show plot

    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'BER vs. Eb/No for {M}-FSK Systems')

    plt.xlabel('Eb/No (dB)')

    plt.ylabel('Bit Error Rate (BER)')

    plt.grid(True, which='both')

    plt.legend()

    plt.show()

# Dropdown for M-FSK selection

bps\_dropdown = Dropdown(

    options=[('2-FSK', 1), ('4-FSK', 2), ('8-FSK', 3), ('16-FSK', 4), ('32-FSK', 5)],

    value=2,

    description='M-FSK:',

)

# Interactive plot

interactive\_plot = interactive(update\_plot, bps=bps\_dropdown)

input\_widgets = VBox([bps\_dropdown], layout=Layout(width='auto'))

plot\_output = interactive\_plot.children[-1]  # The output plot

# Create a VBox that includes both the input widgets and the loading animation

inputs\_and\_loader = HBox([input\_widgets], layout=Layout(align\_items='center'))

# Create an HBox to hold everything in a horizontal layout

ui = VBox([inputs\_and\_loader, plot\_output])

# Display the UI components

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

display(ui)

**Επιστροφή στην υλοποίηση**

#### Ανάλυση Φάσματος coherent και non coherent FSK:

Διαδραστικό γράφημα ανάλυσης της φασματικής πυκνότητας ισχύος τόσο για coherent όσο και για non coherent FSK συστήματα, όπου οι χρήστες μπορούν να προσαρμόσουν τις παραμέτρους όπως ο αριθμός συμβόλων και το Eb/No.

**Κώδικας – Chapter 6 - Analysis coherent και non coherent FSK**

import numpy as np

import matplotlib.pyplot as plt

import ipywidgets as widgets

from ipywidgets import IntSlider, HBox, VBox, Layout, Dropdown, interactive

from IPython.display import display, clear\_output

from scipy.signal import welch

import math

# Define the function to be called when widgets change

def update\_plot(fsklvl, Nsymb, EbNo):

    clear\_output(wait=True)

    bps = int(math.log2(fsklvl))

    # Derived parameters for coherent FSK

    M = 2 \*\* bps  # number of different symbols

    BR = 1  # Baud Rate

    T = 1 / BR  # one symbol period

    fc = 2 \* M \* BR  # RF frequency

    ns = 80  # samples per symbol

    nb = bps \* Nsymb  # number of simulated data bits

    f = fc + (BR / 2) \* ((np.arange(1, M + 1)) - (M + 1) / 2)

    # Calculate the maximum frequency

    fmax = np.max(f)

    # Recalculate ns to ensure Fs = ns \* BR > 2 \* fmax

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    # Recalculate the oversampling period

    Ts = T / ns

    # awgn channel

    SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)  # in dB

    # input data bits

    y = np.random.randint(0, 2, nb)

    x = np.reshape(y, (len(y) // bps, bps))

    # time vectors

    t = np.arange(0, len(x) \* T, T)  # time vector on the T grid

    tks = np.arange(0, T, Ts)  # oversampling time vector

    # FSK signal generation

    s = []

    A = np.sqrt(2 / T / ns)

    for k in range(len(x)):

        fk = f[int(''.join(map(str, x[k])), 2)]

        tk = (k \* T) + tks

        s.append(np.sin(2 \* np.pi \* fk \* tk))

    s\_coherent = np.concatenate(s)

    # Plotting power spectral density for coherent FSK

    frequencies\_coherent, Pxx\_coherent = welch(s\_coherent, fs=ns\*BR, nperseg=50000, noverlap=25000)

    # Derived parameters for non-coherent FSK

    ns = 90  # samples per symbol

    nb = bps \* Nsymb  # number of simulated data bits

    T = 1 / BR  # one symbol period

    Ts = T / ns  # oversampling period

    # Frequencies in "non-coherent" distance

    f = fc + BR \* ((np.arange(1, M + 1)) - (M + 1) / 2)

    # Calculate the maximum frequency

    fmax = np.max(f)

    # Recalculate ns to ensure Fs = ns \* BR > 2 \* fmax

    Fs = 2 \* fmax

    ns = int(np.ceil(Fs / BR)) + 10

    # Recalculate the oversampling period

    Ts = T / ns

    # awgn channel

    SNR = EbNo + 10 \* np.log10(bps) - 10 \* np.log10(ns / 2)  # in dB

    # input data bits

    y = np.random.randint(0, 2, nb)

    x = np.reshape(y, (len(y) // bps, bps))

    # time vectors

    t = np.arange(0, len(x) \* T, T)  # time vector on the T grid

    tks = np.arange(0, T, Ts)  # oversampling time vector

    # FSK signal generation

    s = []

    A = np.sqrt(2 / T / ns)

    for k in range(len(x)):

        fk = f[int(''.join(map(str, x[k])), 2)]

        tk = (k \* T) + tks

        s.append(np.sin(2 \* np.pi \* fk \* tk))

    s\_non\_coherent = np.concatenate(s)

    # Plotting power spectral density for non-coherent FSK

    frequencies\_non\_coherent, Pxx\_non\_coherent = welch(s\_non\_coherent, fs=ns\*BR, nperseg=50000)

    # Plot both side by side

    fig, axs = plt.subplots(2, 1, figsize=(10, 8))

    axs[0].semilogy(frequencies\_coherent, Pxx\_coherent, linewidth=0.5)

    axs[0].set\_title("FSK Coherent")

    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()

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='Nsymb', 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='EbNo', style={'description\_width': 'initial'}, layout=Layout(flex='1 1 auto', width='auto'), continuous\_update=False)

# Create interactive interface

interactive\_plot = interactive(update\_plot, fsklvl=fsklvl\_dropdown, Nsymb=Nsymb\_slider, EbNo=EbNo\_slider)

input\_widgets = VBox([fsklvl\_dropdown, Nsymb\_slider, EbNo\_slider], layout=Layout(flex='1 1 auto', width='auto'))

plot\_output = interactive\_plot.children[-1]  # The output plot

# Create a VBox that includes both the input widgets and the loading animation

inputs\_and\_loader = HBox([input\_widgets])

# Create an HBox to hold everything in a horizontal layout

ui = VBox([inputs\_and\_loader, plot\_output])

# Display the UI components

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

display(ui)

**Επιστροφή στην υλοποίηση**

#### MSK Bit Error Rate:

Διαδραστικό γράφημα για σύγκριση μεταξύ των καμπυλών BER με και χωρίς precoding για τη διαμόρφωση MSK. Οι χρήστες μπορούν να δουν τις θεωρητικές και πειραματικές καμπύλες, συγκρίνοντας τις επιδόσεις για διαφορετικές τιμές Eb/No.

**Κώδικας – Chapter 6 - MSK Bit Error Rate**

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import upfirdn, firwin2, lfilter, welch

from scipy.special import erfc

from numpy import sqrt

import random

# Define AWGN function to simulate noise

def awgn(signal, SNR):

    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

# The MSK error function remains the same

def msk\_errors(Nbits, nsamp, EbNo):

    n = Nbits  # number of data bits

    R = 2000000  # bit rate

    fc = 8000000  # carrier frequency

    ns = nsamp  # oversampling factor

    # AWGN channel

    SNR = EbNo - 10 \* np.log10(ns/2)  # in dB

    T = 1 / R  # 1-bit period (= basic period)

    Ts = T / ns  # sampling frequency

    fss = 1/Ts

    # Input sequence

    y = np.concatenate(([1], np.sign(np.random.rand(n - 1) - 0.5)))  # random numbers, -1 or 1

    x = y

    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]

    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

# The MSK error function remains the same

def msk\_errors\_precoding(Nbits, nsamp, EbNo):

    n = Nbits  # number of data bits

    R = 2000000  # bit rate

    fc = 8000000  # carrier frequency

    ns = nsamp  # oversampling factor

    # AWGN channel

    SNR = EbNo - 10 \* np.log10(ns/2)  # in dB

    T = 1 / R  # 1-bit period (= basic period)

    Ts = T / ns  # sampling frequency

    fss = 1/Ts

    # Input sequence

    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

    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

# Simulate BER for different Eb/No values and plot only the dots for simulation

EbNo\_range = np.arange(0, 9, 1)  # EbNo from 0 to 10 dB

Nbits = 10000  # Increase number of bits to reduce variance

nsamp = 32

simulated\_BER = []

simulated\_BER\_precoding=[]

theoretical\_BER = []

theoretical\_BER\_precoded = []

for EbNo in EbNo\_range:

    sim\_BER = msk\_errors(Nbits, nsamp, EbNo)

    sim\_BER1= msk\_errors\_precoding(Nbits, nsamp, EbNo)

    theoretical\_BER\_value = 0.9 \* erfc(sqrt(10\*\*(EbNo / 10)))  # without precoding

    theoretical\_BER\_precoded\_value = erfc(sqrt(10\*\*(EbNo / 10))) / 2  # with precoding

    simulated\_BER.append(sim\_BER)

    simulated\_BER\_precoding.append(sim\_BER1)

    theoretical\_BER.append(theoretical\_BER\_value)

    theoretical\_BER\_precoded.append(theoretical\_BER\_precoded\_value)

# Plot the results

plt.figure(figsize=(10, 6))

plt.semilogy(EbNo\_range, simulated\_BER, 'bo', label='Simulated BER(without precoding)')  # Dots only for simulation

plt.semilogy(EbNo\_range, simulated\_BER\_precoding, 'ro', label='Simulated BER (with precoding)')  # Dots only for simulation

plt.semilogy(EbNo\_range, theoretical\_BER, 'r-', label='Theoretical BER (without precoding)')

plt.semilogy(EbNo\_range, theoretical\_BER\_precoded, 'g-', label='Theoretical BER (with precoding)')

plt.xlabel('$E\_b/N\_0$ (dB)')

plt.ylabel('Bit Error Rate (BER)')

plt.title('BER vs $E\_b/N\_0$ for MSK Modulation')

plt.legend()

plt.grid(True, which='both')

plt.show()

**Επιστροφή στην υλοποίηση**

## Custom JavaScript

**Κώδικας – Custom Javascript**

/\*\*

 \* Checks if the current page is the target page by analyzing the URL path.

 \* This function helps in ensuring that the custom logic only runs on the specified page.

 \* The page segment '/content/Digcom\_Lab' is checked in the URL path.

 \* This also accounts for potential variable base paths, which may change dynamically.

 \*

 \* @returns {boolean} - True if the current page matches the target path, otherwise false.

 \*/

function isTargetPage() {

    // Get the current page path from the window object

    const urlPath = window.location.pathname;

    // Check if the path includes the specified page segment

    return urlPath.includes('/content/Digcom\_Lab');

}

/\*\*

 \* Hides code cells marked with a specific tag initially.

 \* This function targets elements with IDs that start with 'codecell', within elements tagged '.tag\_hide-code-cell'.

 \* The style is set to 'none' to hide the matched elements, simulating a "collapse" behavior for code cells.

 \* This is useful for controlling the visibility of specific cells at the page load time.

 \*/

function hideInitialCodecells() {

    // Query and loop through all elements matching the hide-code-cell tag and starting with 'codecell'

    document.querySelectorAll('.tag\_hide-code-cell [id^="codecell"]').forEach(element => {

        // Hide each element by changing its display property

        element.style.display = 'none';

        // Log a message to the console for debugging purposes, indicating the element's ID

        console.log(`Element with ID ${element.id} has been initially hidden.`);

    });

}

/\*\*

 \* Custom function to hide input fields within thebelab-executed code cells.

 \* Specifically targets elements that have '.thebelab-input' class, which is used to capture user input in live code cells.

 \* This function hides these inputs after initial page load and during specific status changes.

 \* It acts as a visual enhancement by cleaning up the interface once cells are ready to run.

 \*/

function customFunction() {

    // Query all code cells that match the hide-code-cell tag and start with 'codecell'

    document.querySelectorAll('.tag\_hide-code-cell [id^="codecell"]').forEach(element => {

        // Look for the input field within each matched code cell (class '.thebelab-input')

        let inputElement = element.querySelector('.thebelab-input');

        // If such an input field exists, hide it

        if (inputElement) {

            inputElement.style.display = 'none';

            // Log a message to the console for tracking, indicating the element's ID

            console.log(`Thebelab input within ${element.id} has been hidden.`);

        }

    });

}

/\*\*

 \* Observes DOM mutations to detect when relevant code cells are "ready" for execution.

 \* This function is critical for monitoring changes in the document, especially for dynamic content loaded asynchronously.

 \* It specifically checks for the 'launching' status of Thebelab code cells and triggers the customFunction when the cells are ready.

 \*/

function observeStatusChanges() {

    // Only proceed if the current page matches the target path

    if (!isTargetPage()) return; // Exit early if not on the target page

    // Define the configuration for the observer: we watch for childList changes and text content modifications

    const config = {

        childList: true,

        subtree: true,

        characterData: true,

        attributes: false

    };

    // Define the callback function for the observer, which processes detected mutations

    const callback = function(mutationsList, observer) {

        for (const mutation of mutationsList) {

            // We are interested in new child elements or changes in character data (text content)

            if (mutation.type === 'childList' || mutation.type === 'characterData') {

                // Query elements with the class 'launch\_msg', which indicate a loading or status message

                let elements = document.querySelectorAll('.launch\_msg');

                // Loop through each status message element

                elements.forEach((element) => {

                    // Check if the text indicates a ready state (e.g., launching from mybinder)

                    if (element.textContent === 'Launching from mybinder.org: ') {

                        // When the status is ready, invoke customFunction to hide thebelab input elements

                        customFunction();

                    }

                });

            }

        }

    };

    // Create a new MutationObserver instance, passing in the callback function defined above

    const observer = new MutationObserver(callback);

    // Begin observing the entire document body with the specified configuration

    observer.observe(document.body, config);

}

/\*\*

 \* Main initialization logic.

 \* This function ensures that the custom logic only runs when the target page is loaded and the DOM is fully parsed.

 \* It first hides the code cells and then sets up the mutation observer to track status changes.

 \*/

if (isTargetPage()) {

    document.addEventListener('DOMContentLoaded', (event) => {

        // Log a message to indicate that the custom script is starting

        console.log("starting custom code");

        // Initially hide code cells on page load

        hideInitialCodecells();

        // Set up an observer to track status changes and respond to them

        observeStatusChanges();

    });

}

**Επιστροφή στην υλοποίηση**