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 ΕΜΠ

Αθήνα, Ιούνιος 2025

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 26η Ιουνίου 2025

.....	.....	.....
Μήτρου Νικόλαος	Σύκας Ευστάθιος	Ρουσσάκη Ιωάννα
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγήτρια ΕΜΠ

Αθήνα, Ιούνιος 2025

.....
Αραβαντινός-Καρλάτος Γ. Σωτήρης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών ΕΜΠ

Περίληψη

Η παρούσα διπλωματική εργασία εστιάζεται στην ανάπτυξη μιας web πλατφόρμας με διαδραστικές δυνατότητες για προσομοίωση συστημάτων ψηφιακής επικοινωνίας μέσω χρήσης ελεύθερου λογισμικού. Η εργασία συνοψίζει βασικά θεωρητικά στοιχεία από τις σημειώσεις του μαθήματος «Ψηφιακές Επικοινωνίες Ι», και προχωρεί σε αναπαραστάσεις και προσομοιώσεις των βασικών διατάξεων και συστημάτων που αφορούν την ψηφιακή επεξεργασία σήματος, τις τεχνικές ανίχνευσης και διαμόρφωσης, όπως και τα χαρακτηριστικά του φάσματος των ψηφιακών κυματομορφών.

Η εργασία περιλαμβάνει επισκόπηση 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 την 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 enable 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 functions 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 simulation outcomes and their comparison with theoretical results, 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, καθώς επίσης, και στα μέλη της ΚΟΥ ΚΑΛΛΙΠΟΣ, Γεωργία Τριανταφυλλίδου, Ηλία Τσιώνη και Χρήστο Κεντρωτή για τη βοήθεια στη γλωσσική και τεχνική επιμέλεια των τευχών.

Η παρούσα διπλωματική εργασία σηματοδοτεί την ολοκλήρωση των προπτυχιακών μου σπουδών στη Σχολή 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, και ειδικότερα οι στόχοι που τέθηκαν, καθώς και η συνολική δομή της. Στο δεύτερο κεφάλαιο γίνεται μια επισκόπηση της βιβλιογραφίας, με αναφορές στις σημειώσεις του μαθήματος «Ψηφιακές Επικοινωνίες Ι» και στις ενότητες που θα γίνουν διαδραστικές, καλύπτοντας θέματα όπως την ψηφιακή επεξεργασία σήματος, την ανίχνευση μέσω προσαρμοσμένων φίλτρων και τις 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 την 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

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Στόχοι της Εργασίας

Οι στόχοι της παρούσας εργασίας επικεντρώνονται στην αναζήτηση, στην αξιολόγηση, στην επιλογή και στη 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 ανοικτού 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.

1.2 Δομή της Εργασίας

Η δομή της παρούσας εργασίας αντανακλά μια οργανωμένη και συστηματική προσέγγιση της έρευνας, της αξιολόγησης και της 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, παρουσιάζονται η έρευνα και η αξιολόγηση των διαθέσιμων εργαλείων ανοικτού κώδικα, που θα μπορούσαν να χρησιμοποιηθούν για την ανάπτυξη της πλατφόρμας. 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

Επισκόπηση της Βιβλιογραφίας

2.1 Στοιχεία θεωρίας Ψηφιακών Επικοινωνιών

Η ενότητα αυτή παρουσιάζει συνοπτικά τα βασικά στοιχεία της θεωρίας που αφορούν τις σύγχρονες Ψηφιακές Επικοινωνίες, βάσει των σημειώσεων του μαθήματος «Ψηφιακές Επικοινωνίες Ι» [1] (ΣΗΜΜΥ, 6ο εξάμηνο). Ειδικότερα επισημαίνονται τα τμήματα θεωρίας για τα οποία θα πρέπει να αναπτυχθούν εργαλεία προσομοίωσής τους με διαδραστικό τρόπο, με βάση τις αντίστοιχες ερωτήσεις στις εργαστηριακές ασκήσεις.

Τα εργαλεία 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 προσαρμοσμένων φίλτρων, τα χαρακτηριστικά φάσματος ψηφιακών κυματομορφών, καθώς και οι διάφορες τεχνικές διαμόρφωσης ASK, QAM, PSK, FSK και MSK.

2.1.1 Η Ψηφιακή Επεξεργασία Σήματος στις Τηλεπικοινωνίες [2]

«Το κεφάλαιο αυτό περιγράφει 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 στην 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.»

Τα στοιχεία που θα υλοποιηθούν με διαδραστικά εργαλεία είναι τα ακόλουθα:

- **Δημιουργία & Οπτικοποίηση σημάτων:** Παραγωγή σημάτων ψηφιακών επικοινωνιών και δημιουργία γραφημάτων για να απεικονιστούν τα χρονικά και φασματικά διαγράμματα αυτών.

- **Διαδραστική ρύθμιση μεταβλητών σήματος:** Με τη χρήση διαδραστικών εργαλείων, να μπορούν οι χρήστες να τροποποιούν τις παραμέτρους των σημάτων 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 χρονικής απόκρισής τους σε μορφή stem plot.

- **Απόκριση Συχνότητας Φίλτρων (Filter Frequency Response):** Διαδραστική απεικόνιση της φασματικής απόκρισης φίλτρων για διαφορετικά μήκη χρονικής απόκρισης (FIR) και παράθυρα.

- **Ανάλυση Equiripple Φίλτρων:** Δυνατότητα επιλογής equiripple φίλτρων διαφορετικών μηκών και σύγκριση της φασματικής τους απόκρισης.

- **Εφαρμογή Βαθυπερατών Φίλτρων (Low Pass Filter Application):** Εφαρμογή βαθυπερατών φίλτρων και απεικόνιση της Φασματικής Πυκνότητας Ισχύος (Power Spectral Density) του φιλτραρισμένου σήματος.

- **Ανάλυση Απόκρισης Bandpass Φίλτρων:** Εφαρμογή φίλτρων bandpass και οπτική απεικόνιση της απόκρισής τους τόσο στη συχνότητα όσο και στη φασματική πυκνότητα του φιλτραρισμένου σήματος.

- **Εφαρμογή ζωνοπερατού Φίλτρου Parks-McClellan:** 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, με απεικόνιση της φασματικής του απόκρισης.

Βέλτιστη ψηφιακή αναγνώριση – Προσαρμοσμένα φίλτρα [3]

Το κεφάλαιο εξετάζει 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 διαφορετικούς, γραμμικά ανεξάρτητους παλμούς του συμφωνημένου ρεπερτορίου μετάδοσης - γνωστού ως «σηματικός αστερισμός») 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 τη μεγαλύτερη πιθανότητα εκπομπής (δηλαδή, τη 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).

Τα στοιχεία που θα υλοποιηθούν διαδραστικά είναι τα ακόλουθα:

- **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 ιστογράμματος για διάφορες τιμές.
- **ASK Bit Error Rate (BER) Visualization:** Η διαδραστική δυνατότητα σύγκρισης διαμορφώσεων M-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 παλμούς ορθογωνικούς και ημιτονοειδείς.

Φασματικά χαρακτηριστικά – Φίλτρα Nyquist [4]

Στόχος αυτού του κεφαλαίου είναι η ανάλυση των χαρακτηριστικών φάσματος των ψηφιακών κυματομορφών, καθώς και η παρουσίαση της σηματοδοσίας Nyquist, η οποία χρησιμοποιείται για την ελαχιστοποίηση των παραμορφώσεων λόγω της διασποράς και για 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 σηματοδοσία Nyquist, εστιάζοντας στη διασφάλιση της μέγιστης απόδοσης των σημάτων χωρίς διασυμβολική παρεμβολή.

Τα στοιχεία που θα υλοποιηθούν διαδραστικά είναι τα ακόλουθα:

- **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.
- **Καμπύλες BER:** 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.

2.1.2 Ψηφιακή Διαμόρφωση QAM και PSK [5]

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. Οι στόχοι 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 την ανάλυση σημάτων QAM και PSK, καθώς και την κατανόηση των επιπτώσεων του θορύβου και του φάσματος σε αυτά τα συστήματα διαμόρφωσης.

Τα στοιχεία που θα υλοποιηθούν διαδραστικά είναι τα ακόλουθα:

- **QAM και PSK Constellations:** Διαδραστικά διαγράμματα αστερισμών για L-QAM και M-PSK. Οι χρήστες θα μπορούν να επιλέγουν την τάξη διαμόρφωσης και να βλέπουν τις αντίστοιχες θέσεις των συμβόλων στον αστερισμό.

- **QAM και PSK Καμπύλες BER:** Δημιουργία γραφημάτων BER τόσο για QAM όσο και για PSK διαμορφώσεις, όπου οι χρήστες θα μπορούν να συγκρίνουν τη 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 τιμές συχνοτήτων f1 και f2.

- **Διαδραστικός Υπολογισμός Παραμέτρων:** Δυνατότητα υπολογισμού παραγόμενων παραμέτρων, με βάση τις βασικές παραμέτρους (R, M και roll-off) που επιλέγει ο χρήστης.

2.1.3 Ψηφιακή διαμόρφωση FSK και MSK [6]

Το κεφάλαιο αυτό επικεντρώνεται στις τεχνικές ψηφιακής διαμόρφωσης FSK (Διαμόρφωση Μετατόπισης Συχνότητας) και MSK (Ελάχιστης Μετατόπισης Συχνότητας). 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 σύμφωνης 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.

Τα στοιχεία που θα υλοποιηθούν διαδραστικά είναι τα ακόλουθα:

- **Καμπύλες BER συναρτήσει του 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 ως συνάρτηση του Eb/No για διαμόρφωση FSK.

- **BER για σύμφωνη και ασύμφωνη FSK:** Διαδραστική σύγκριση των καμπυλών BER για coherent και non coherent συστήματα FSK. Οι χρήστες μπορούν να δουν τις καμπύλες που αντιπροσωπεύουν τη 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 τον αριθμό συμβόλων και το Eb/No.

- **Καμπύλες BER MSK:** 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.

2.2 Διερεύνηση Εργαλείων Υλοποίησης

2.2.1 Εισαγωγή

Gitub

Για την οργάνωση του γενικότερου πλαισίου του έργου, την αποθήκευση των αρχείων και την τελική ανάπτυξη της διαδικτυακής εφαρμογής, χρησιμοποιήθηκε το Gitub [7]. Η πλατφόρμα αυτή επέτρεψε τη διαχείριση του κώδικα και την ασφαλή αποθήκευση των αρχείων του έργου, ενώ μέσω της υπηρεσίας Gitub Pages πραγματοποιήθηκε η τελική ανάπτυξη της εφαρμογής, προσφέροντας έναν εύκολο και αξιόπιστο τρόπο για τη δημοσίευση του περιεχομένου στο Διαδίκτυο.

2.2.2 Διερεύνηση Εργαλείων

Για την επίτευξη των παραπάνω διαδραστικών δυνατοτήτων θα χρειαστεί να βρεθούν εργαλεία 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.
- Διαδραστικές 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. [8]

2. **Voila:** Μετατρέπει Jupyter Notebooks σε αυτόνομες διαδραστικές web εφαρμογές χωρίς την ανάγκη γραφής JavaScript. [9]

3. **MkDocs:** Στατικός site generator που μετατρέπει αρχεία Markdown σε όμορφα τεκμηριωμένες ιστοσελίδες. **[10]**

4. **Dash:** Πλαίσιο για τη δημιουργία διαδραστικών web εφαρμογών για ανάλυση δεδομένων με Python. **[11]**

5. **Google Colab:** Διαδικτυακό περιβάλλον για την εκτέλεση Jupyter Notebooks, με υποστήριξη για Python και ενσωμάτωση στο Google Drive. **[12]**

Ενσωμάτωση kernel για εκτέλεση κώδικα σε πραγματικό χρόνο

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. [13]

2. **binderhub_url:** Παρέχει τη δυνατότητα σύνδεσης με ένα Binderub 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. [14]

3. **jupyterhub_url:** Επιτρέπει τη σύνδεση με ένα Jupyterub server, όπου οι χρήστες μπορούν να εκτελούν τα notebooks σε ένα κοινόχρηστο περιβάλλον εργασίας. [15]

4. **thebe:** Μια JavaScript βιβλιοθήκη που μετατρέπει στατικές σελίδες TML σε διαδραστικές, επιτρέποντας την εκτέλεση κώδικα απευθείας μέσα από την ιστοσελίδα. Το Thebe συνδέεται με έναν Jupyter kernel μέσω ενός backend, όπως το Binder. [16]

5. **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. [17]

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. [18]
2. **SciPy:** Προσφέρει ένα ευρύ φάσμα επιστημονικών και τεχνικών υπολογισμών, συμπεριλαμβανομένων υπολογισμών επεξεργασίας σημάτων και εικόνων. [19]
3. **scipy.signal:** Υπο-πακέτο του SciPy, εξειδικευμένο στην επεξεργασία σημάτων, φίλτρων και συστημάτων. [20]
4. **CommPy:** Βιβλιοθήκη για την προσομοίωση τηλεπικοινωνιακών συστημάτων, συμπεριλαμβανομένων διαμορφώσεων, καναλιών και κωδικοποίησης. [21]
5. **math:** Ενσωματωμένη βιβλιοθήκη της Python με βασικές μαθηματικές συναρτήσεις. [22]
6. **Matplotlib:** Βιβλιοθήκη για γραφική απεικόνιση δεδομένων σε 2D και 3D. [23]
7. **SimPy:** Βιβλιοθήκη για την προσομοίωση διακριτών γεγονότων, χρήσιμη για συστήματα επικοινωνίας. [24]
8. **PyWavelets:** Βιβλιοθήκη για ανάλυση wavelet, χρήσιμη στην επεξεργασία σημάτων. [25]
9. **scikit-dsp-comm:** Παρέχει εργαλεία για επεξεργασία ψηφιακών σημάτων και επικοινωνιών. [26]
10. **PySDR:** Συνοδευτική βιβλιοθήκη για το βιβλίο "Python SDR", με εργαλεία για Software Defined Radio. [27]

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. [23]
2. **Seaborn:** Βιβλιοθήκη που βασίζεται στο Matplotlib και προσφέρει υψηλού επιπέδου διεπαφές για στατιστικά γραφήματα. [28]
3. **Plotly:** Βιβλιοθήκη για διαδραστικά γραφήματα που μπορούν να ενσωματωθούν σε ιστοσελίδες. [29]
4. **Bokeh:** Παρέχει δυνατότητες για διαδραστικά γραφήματα σε ιστοσελίδες μέσω Python. [30]
5. **PyPlot:** Μέρος του Matplotlib. Εύκολη στη χρήση διεπαφή για τη δημιουργία γραφημάτων, παρόμοια με το MATLAB. [31]
6. **ggplot:** Βιβλιοθήκη εμπνευσμένη από το ggplot2 της R, για δηλωτική δημιουργία γραφημάτων. [32]
7. **Altair:** Βιβλιοθήκη για δηλωτική στατιστική οπτικοποίηση βασισμένη στο Vega-Lite. [33]
8. **Pandas Visualization:** Παρέχει απλές μεθόδους για τη δημιουργία γραφημάτων απευθείας από δεδομένα Pandas. [34]
9. **oloviews :** Διευκολύνει τη δημιουργία σύνθετων γραφημάτων με λιγότερο κώδικα. [35]
10. **VisPy:** Βιβλιοθήκη για γρήγορη οπτικοποίηση δεδομένων χρησιμοποιώντας GPU. [36]

Διαδραστικές μέθοδοι εισαγωγής παραμέτρων

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

1. **IPython:** Προηγμένη διεπαφή για την Python που παρέχει διαδραστικό περιβάλλον εργασίας, συμπεριλαμβανομένων δυνατοτήτων όπως τα Jupyter Notebooks. [37]

2. **ipywidgets:** Βιβλιοθήκη που επιτρέπει τη δημιουργία διαδραστικών widget σε Jupyter Notebooks, διευκολύνοντας την εισαγωγή παραμέτρων και την αλληλεπίδραση με τον κώδικα. [38]

3. **Bokeh:** Βιβλιοθήκη για τη δημιουργία διαδραστικών οπτικοποιήσεων σε ιστοσελίδες, υποστηρίζοντας widgets για εισαγωγή παραμέτρων. [30]

4. **Plotly Dash:** Πλατφόρμα για τη δημιουργία διαδραστικών web εφαρμογών με Python, επιτρέποντας την εισαγωγή παραμέτρων μέσω διάφορων στοιχείων διεπαφής. [39]

5. **Streamlit:** Πλαίσιο για τη δημιουργία web εφαρμογών για μηχανική μάθηση και επιστήμη δεδομένων με απλή σύνταξη Python. [40]

6. **Voila:** Επιτρέπει τη μετατροπή Jupyter Notebooks σε διαδραστικές web εφαρμογές χωρίς την ανάγκη γραφής κώδικα JavaScript. [9]

7. **Panel:** Βιβλιοθήκη για τη δημιουργία διαδραστικών dashboards και εφαρμογών, υποστηρίζοντας πολλαπλές βιβλιοθήκες γραφικών. [41]

8. **Gradio:** Επιτρέπει τη δημιουργία απλών διεπαφών χρήστη για μηχανική μάθηση και επιστήμη δεδομένων. [42]

9. PySimpleGUI: Παρέχει μια απλή διεπαφή για τη δημιουργία GUI εφαρμογών με Python. [43]

10. Tkinter: Ενσωματωμένη βιβλιοθήκη της Python για τη δημιουργία GUI εφαρμογών. [44]

2.2.3 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 στη 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 τη 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 απαραίτητες 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.

2.2.4 Επιλογή Εργαλείων

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

Διαδραστική πλατφόρμα παρουσίασης του τελικού αποτελέσματος

Jupyter Book **[8]**	

Γιατί το επιλέξαμε:

- **Ενσωμάτωση 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 στο Gitub 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 τη 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 στον 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 για την 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

[16]

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

[14]

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 στα 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 στη 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
   [18]

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.

2. SciPy
   [19]

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.

3. scipy.signal
   [20]

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

Πλεονεκτήματα:

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

4. commpy
   [21]

commpy

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.

5. Math
   [22]

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 τη διαχείριση μεγάλων δεδομένων με υψηλή απόδοση.
2. **scipy.signal:** Επέτρεψε τον σχεδιασμό και την ανάλυση ψηφιακών φίλτρων, απαραίτητων για τις εργαστηριακές ασκήσεις που αφορούν την επεξεργασία σημάτων και την 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 στην 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, στην πλούσια τεκμηρίωσή τους και στη δυνατότητά τους να ικανοποιήσουν τις συγκεκριμένες ανάγκες της εργασίας.

Γραφική αναπαράσταση δεδομένων

Matplotlib

[23]

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 τη 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, στην πλούσια τεκμηρίωση και στη δυνατότητα να ικανοποιήσει τις συγκεκριμένες απαιτήσεις του μαθήματος.

Διαδραστικές μέθοδοι εισαγωγής παραμέτρων

1. IPython
   [37]

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 TML, 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.

2. ipywidgets
   [38]

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 στα 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 ενισχύουν την κατανόηση και τη διατήρηση των γνώσεων.
- **Ευκολία Ανάπτυξης:** Η απλή σύνταξη και η άμεση ενσωμάτωση με το υπάρχον περιβάλλον εργασίας διευκολύνουν 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 τη 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 στη μάθηση των ψηφιακών συστημάτων επικοινωνίας.

Κεφάλαιο 3

Μεθοδολογία, Σχεδίαση και Υλοποίηση

3.1 Εισαγωγή

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

3.2 Μέθοδος υλοποίησης του συνολικού έργου

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, 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 διπλωματική εργασία [45].

1. **Αναζήτηση, δοκιμή και τελική επιλογή των κατάλληλων εργαλείων για την ολοκλήρωση του project.**

2. **Δημιουργία αποθετηρίου στο Gitub για τη διαχείριση των αρχείων του έργου.**

3. **Κατασκευή βασικών τμημάτων κώδικα python γύρω από τη 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 εργαλείου 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 Gitub Pages για την ανάπτυξη και τη φιλοξενία της διαδικτυακής εφαρμογής.**

9. **Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για τη δημοσίευση της ιστοσελίδας στο Gitub 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 υπάρχοντα 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 τη βοήθεια των κομματιών κώδικα από το βήμα 3)

15. Προσθήκη διαδραστικών λειτουργιών στον κώδικα Python μέσω των κατάλληλων βιβλιοθηκών. (Με τη 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 τη δημοσίευση της ιστοσελίδας στο Gitub Page που δημιουργήσαμε.

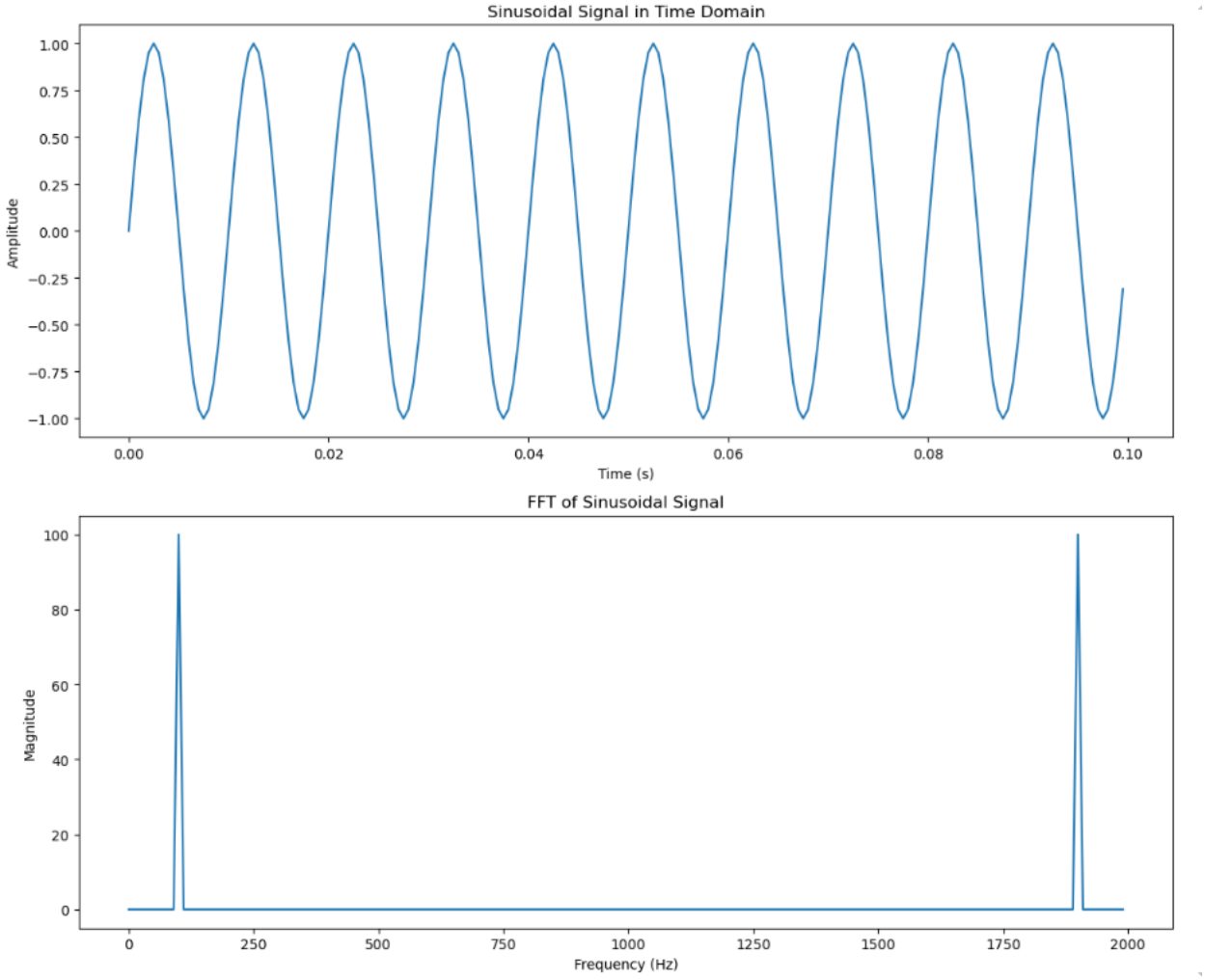
3.3 Υλοποίηση διαδραστικών στοιχείων

3.3.1 Ψηφιακή Επεξεργασία Σήματος στις Τηλεπικοινωνίες

3.3.1.1 Δημιουργία & Οπτικοποίηση σημάτων

Παραγωγή σημάτων διακριτού χρόνου 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

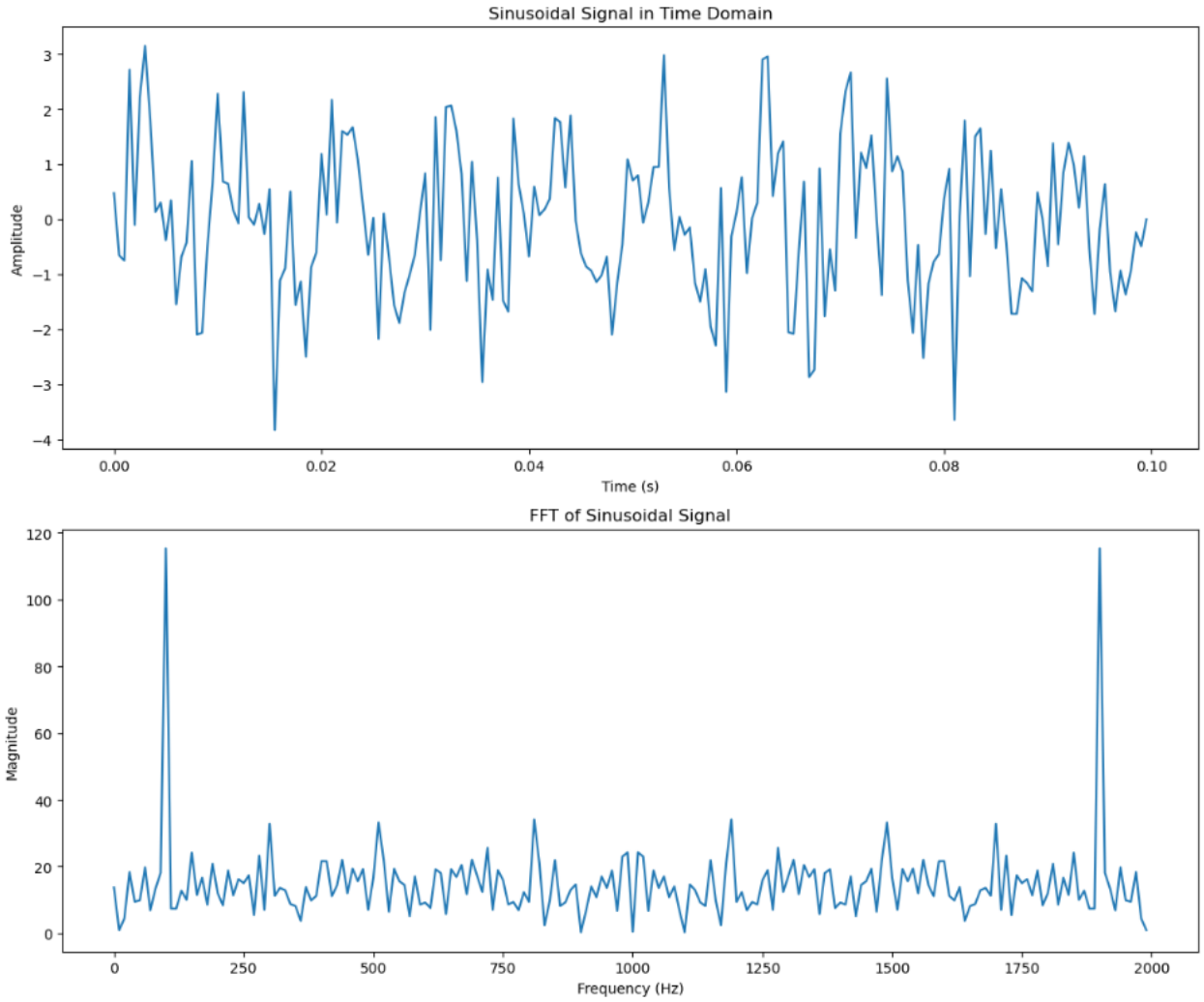


Σχήμα 1: Δημιουργία & Οπτικοποίηση σημάτων

3.3.1.2 Προσθήκη θορύβου

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

Κώδικας – Chapter 1 – Add noise

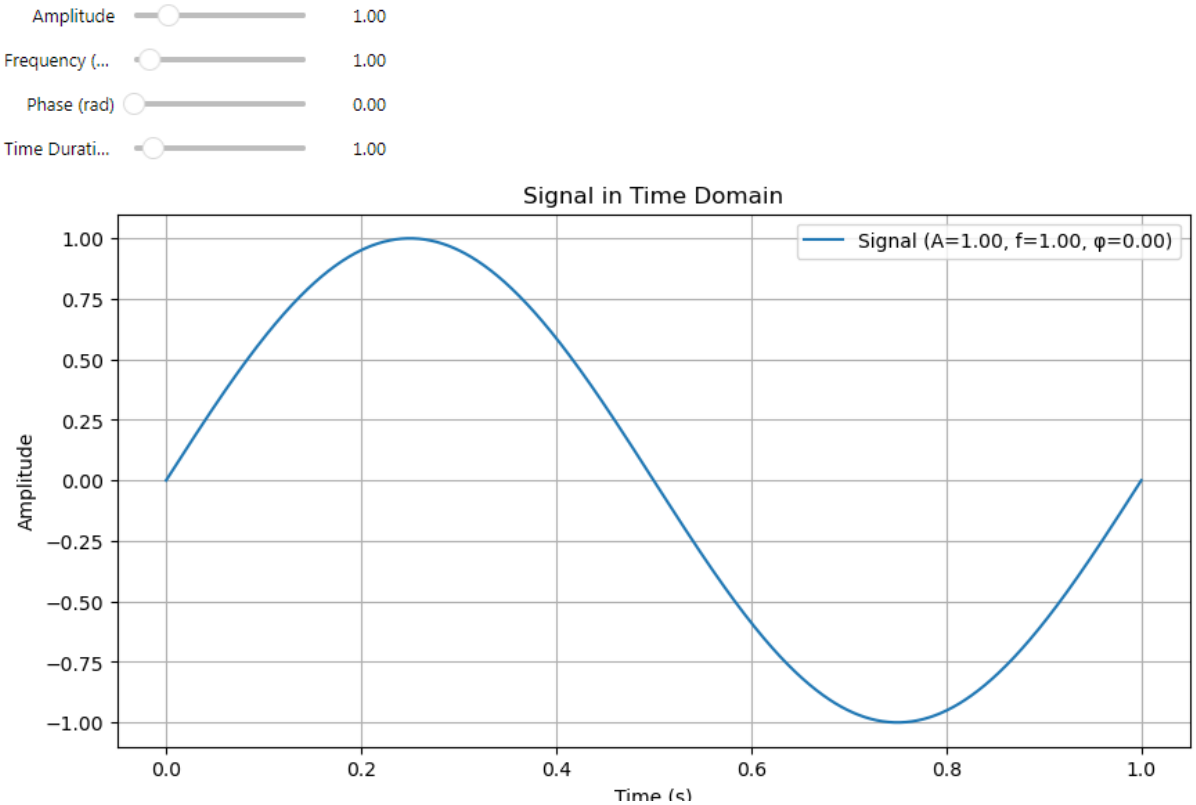


Σχήμα 2: Προσθήκη θορύβου

3.3.1.3 Διαδραστική ρύθιση μεταβλητών σήματος

Με τη χρήση διαδραστικών εργαλείων, οι χρήστες μπορούν να τροποποιούν τις παραμέτρους του σήματος όπως είναι το πλάτος, η συχνότητα και η διάρκεια, και να παρατηρούν σε πραγματικό χρόνο τις αλλαγές.

Κώδικας – Chapter 1 – Interactive Signal

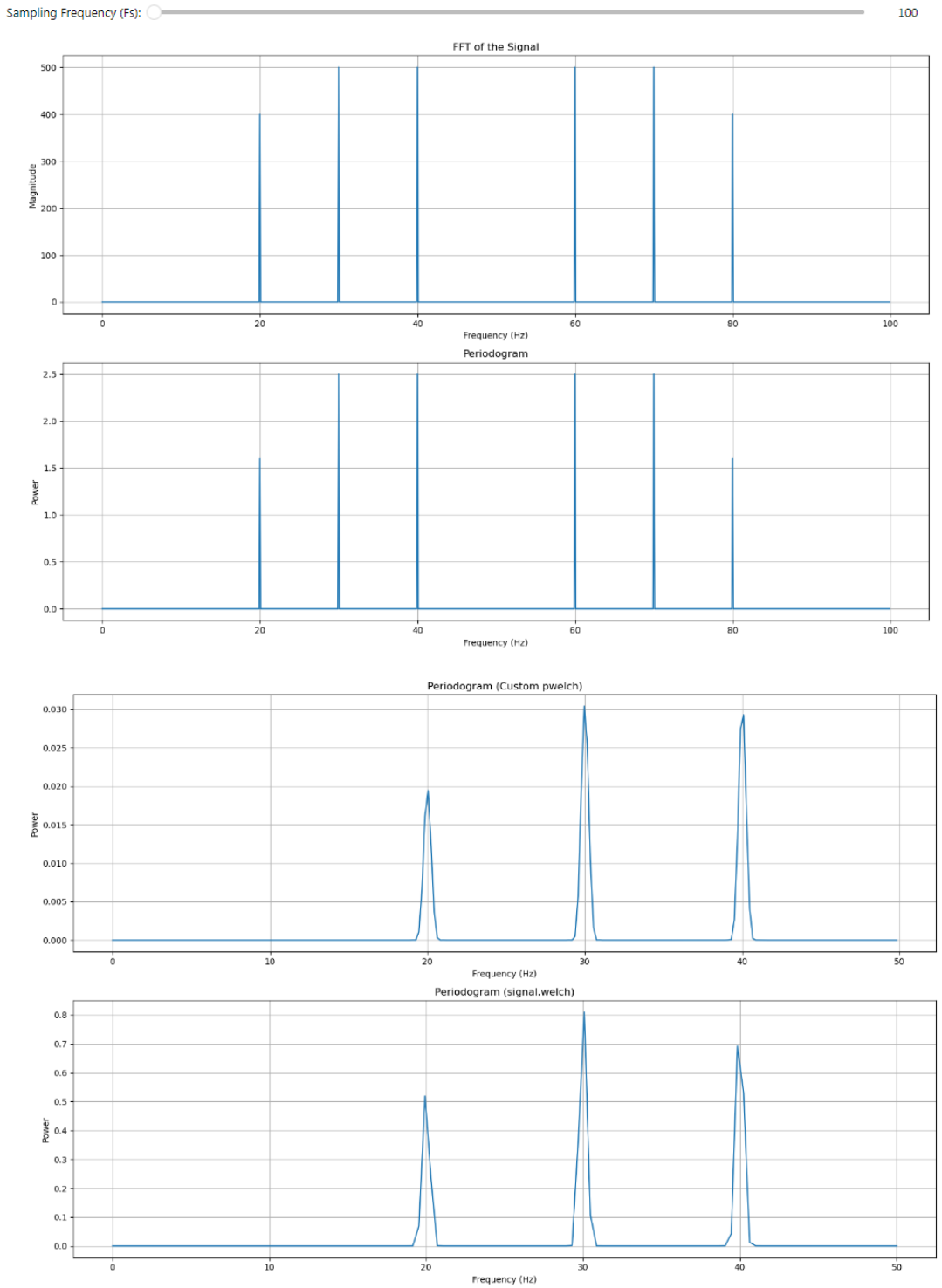


Σχήμα 3: Διαδραστική ρύθιση μεταβλητών σήματος

3.3.1.4 Ανάλυση Φάσματος

Παράδειγμα: Δημιουργία γραφημάτων φασματικής ανάλυσης σήματος με τρεις συχνότητες.

Κώδικας – Chapter 1 - Spectrum Analysis

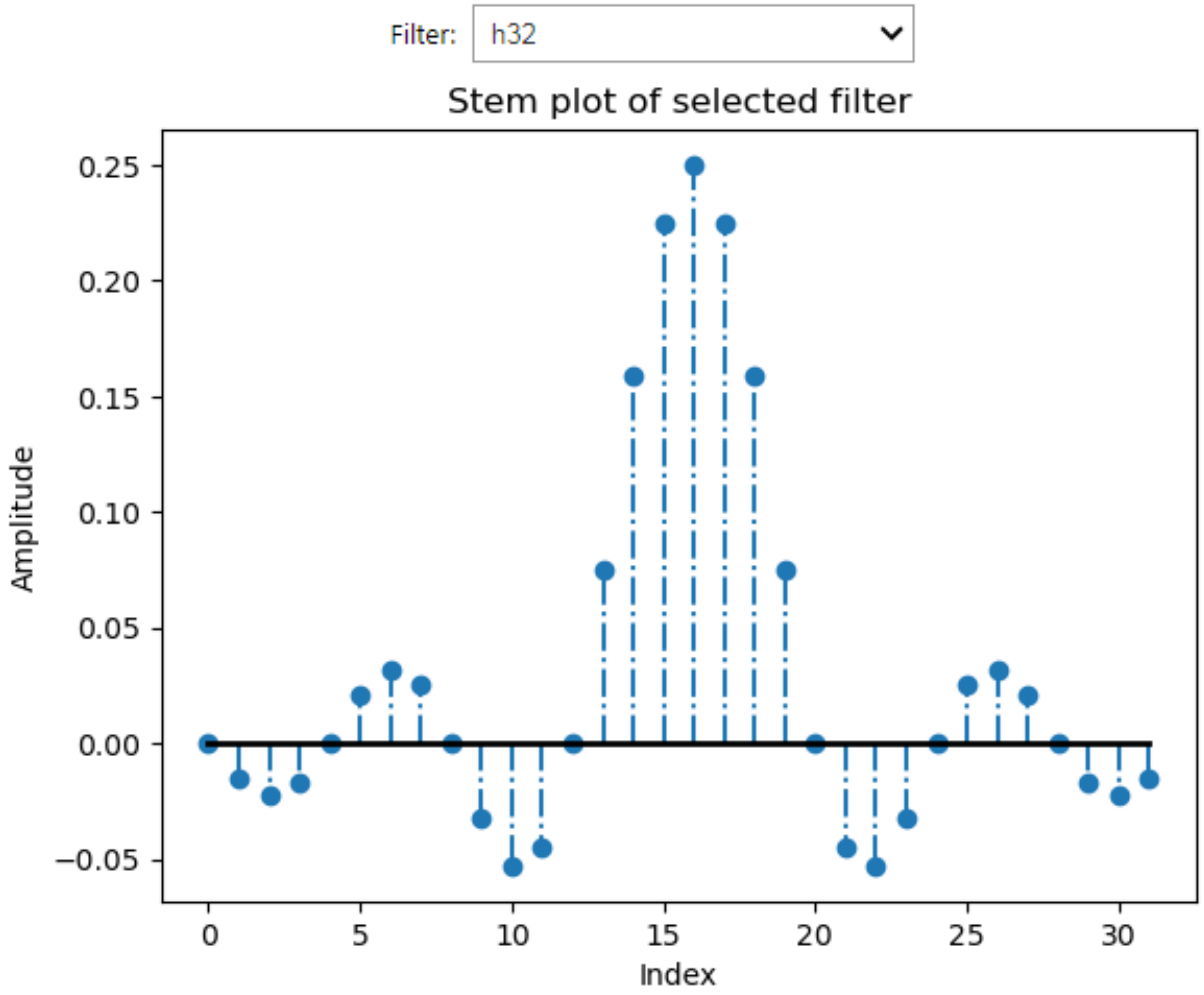


Σχήμα 4: Ανάλυση Φάσματος

3.3.1.5 Απεικόνιση Φίλτρου (Filter Visualization)

Δυνατότητα επιλογής διαφορετικών τύπων και τάξης φίλτρων και οπτική αναπαράσταση της απόκρισής τους με μορφή μίσχων (stem plot).

Κώδικας – Chapter 1 - Filter Visualization

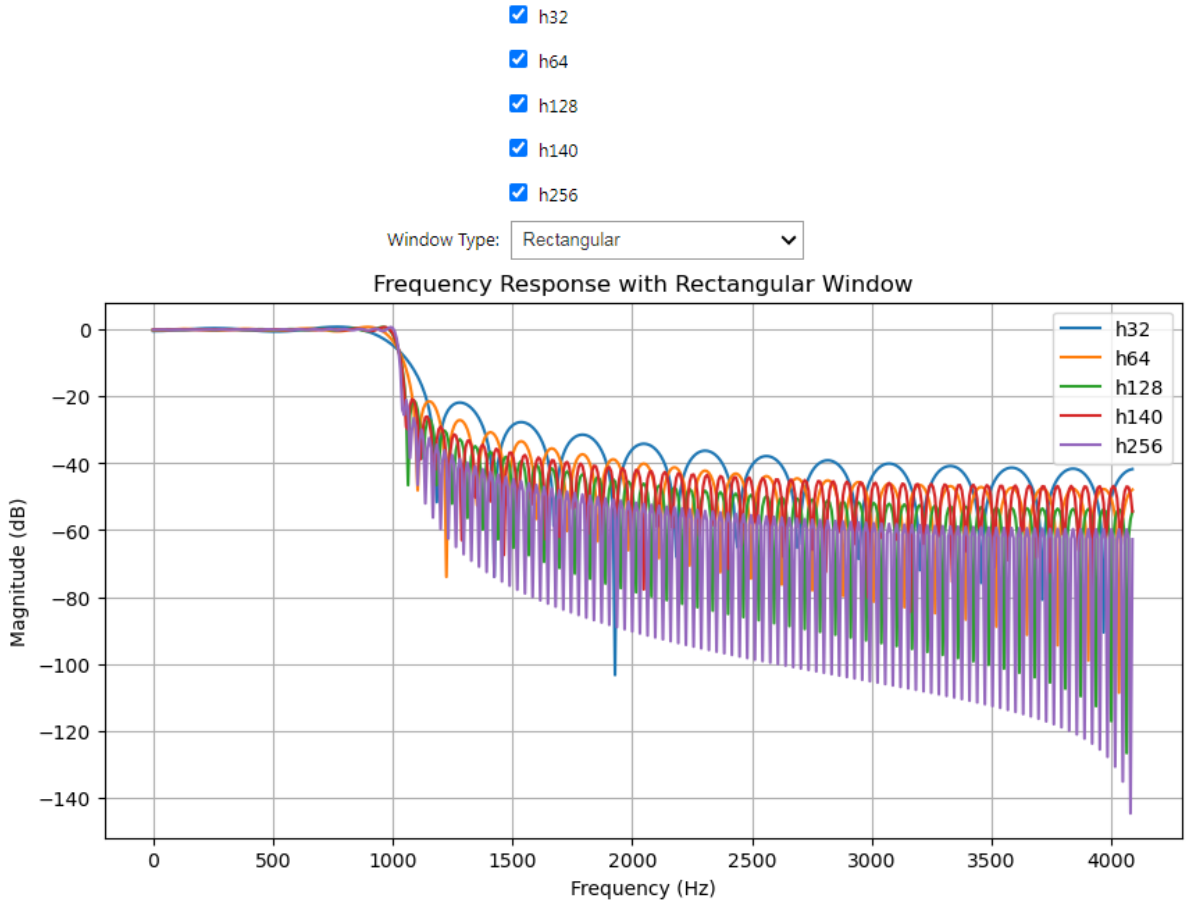


Σχήμα 5: Απεικόνιση Φίλτρου (Filter Visualization)

3.3.1.6 Ανάλυση Συχνοτήτων Φίλτρων (Filter Frequency Response)

Διαδραστική απεικόνιση της φασματικής απόκρισης φίλτρων για διαφορετική τάξη και διαφορετικά παράθυρα.

Κώδικας – Chapter 1 - Filter Frequency Response

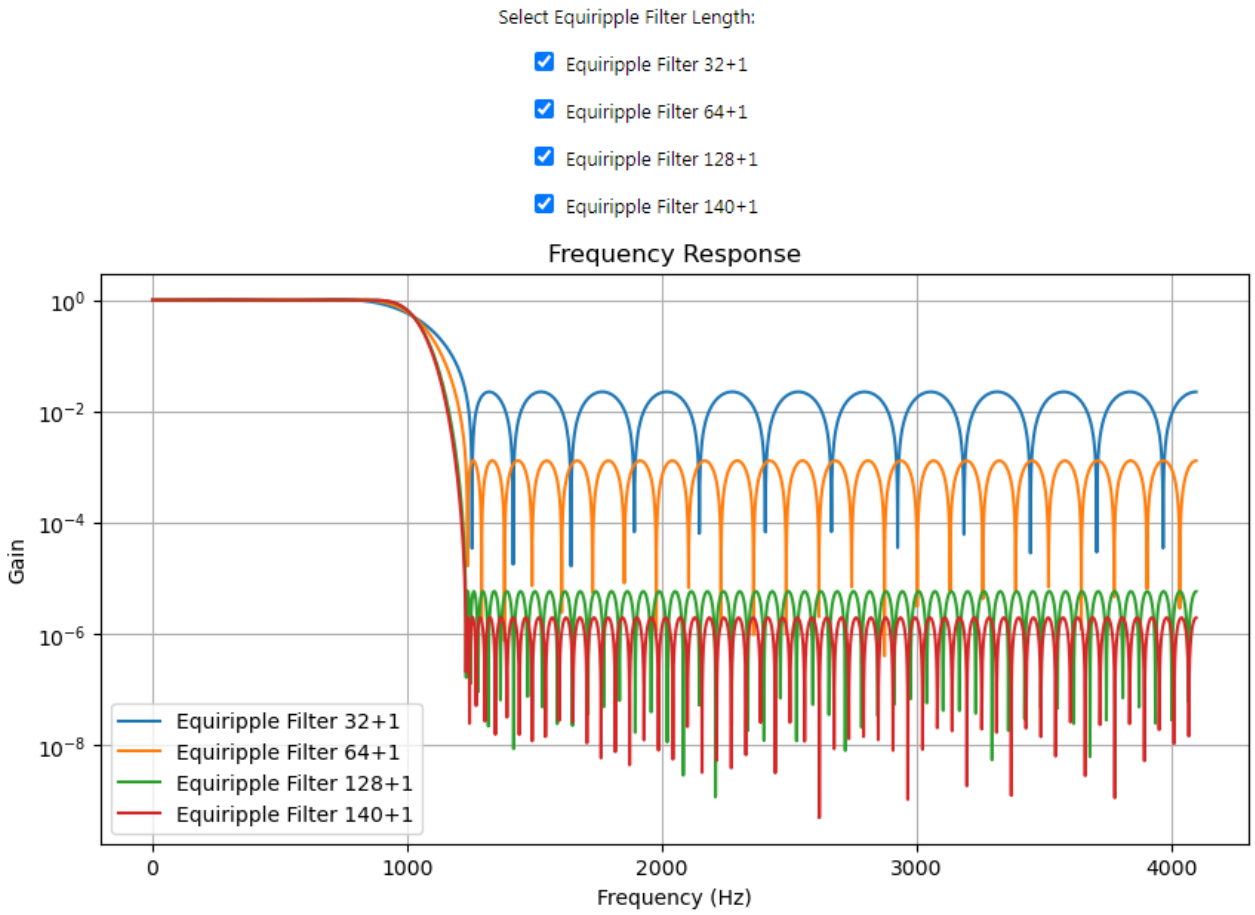


Σχήμα 6: Ανάλυση Συχνοτήτων Φίλτρων (Filter Frequency Response)

3.3.1.7 Ανάλυση Φίλτρων Equiripple

Δυνατότητα επιλογής equiripple φίλτρων Parks-McClellan διαφορετικής τάξης (μήκους κρουστικής απόκρισης) και σύγκριση της φασματικής τους απόκρισης.

Κώδικας – Chapter 1 - Equiripple Filter Frequency Response

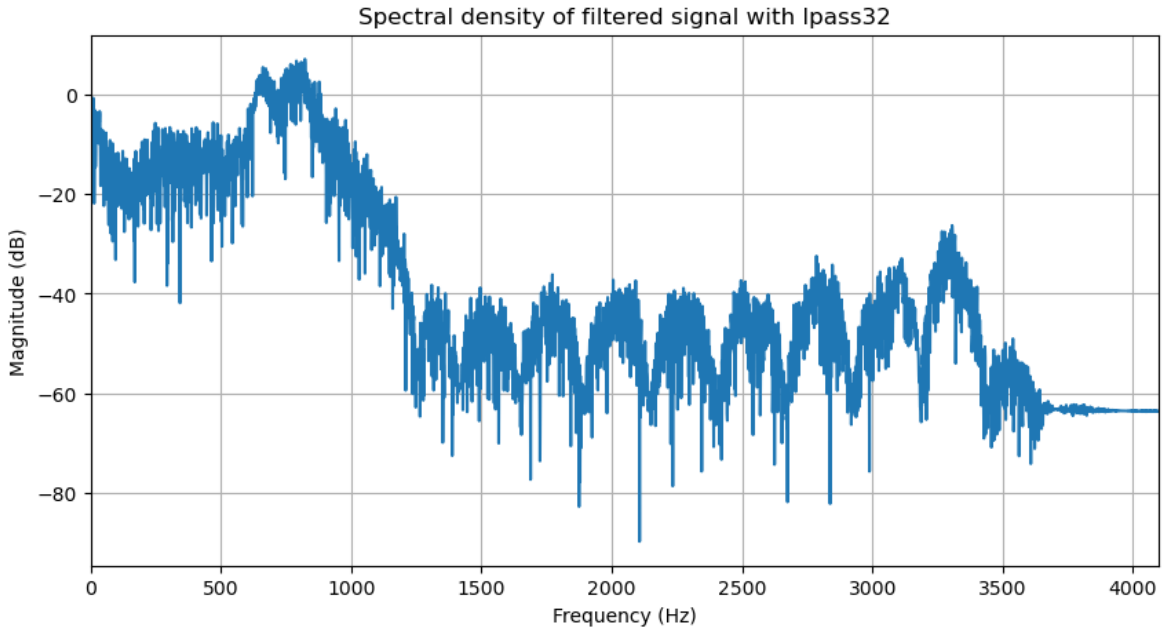


Σχήμα 7: Ανάλυση Φίλτρων Equiripple

3.3.1.8 Εφαρμογή Βαθυπερατού Φίλτρου (Low Pass Filter Application)

Εφαρμογή βαθυπερατών φίλτρων και απεικόνιση της φασματικής πυκνότητας του φιλτραρισμένου σήματος.

Κώδικας – Chapter 1 - Low Pass Filter Application

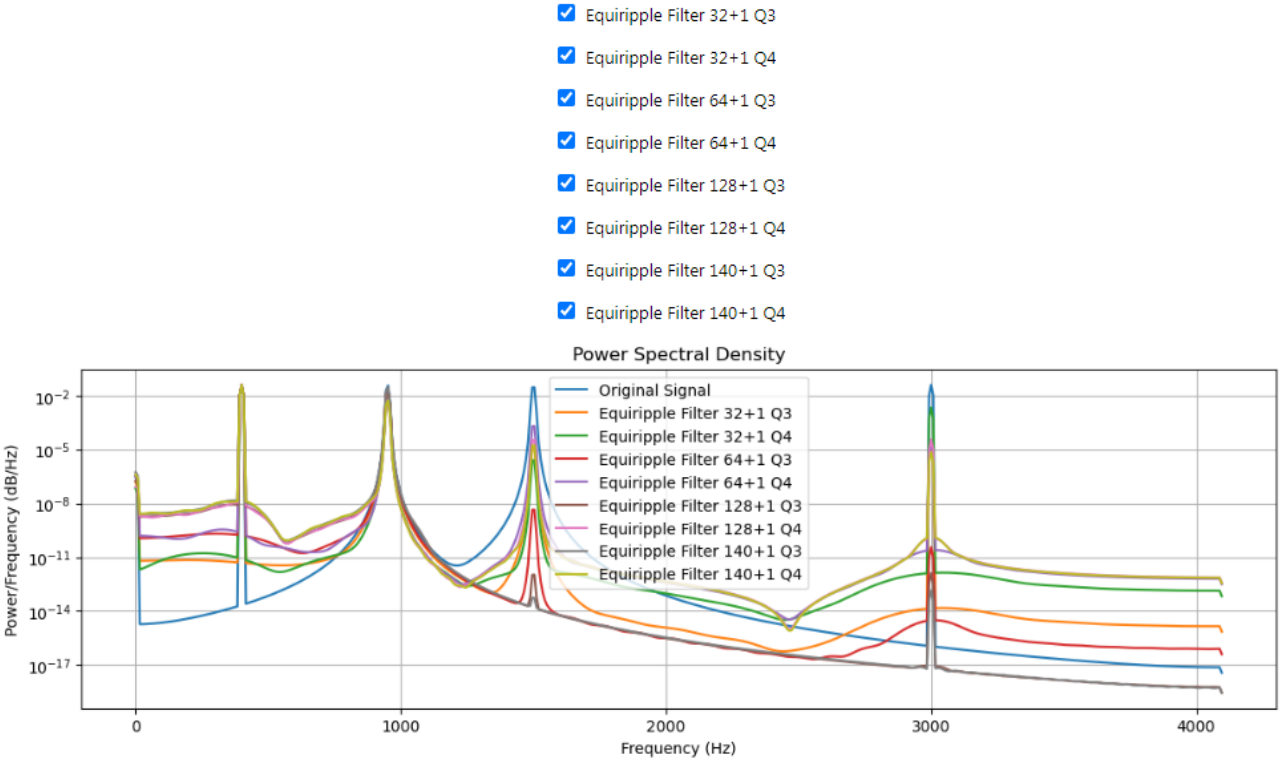


Σχήμα 8: Εφαρμογή Βαθυπερατού Φίλτρου (Low Pass Filter Application)

3.3.1.9 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

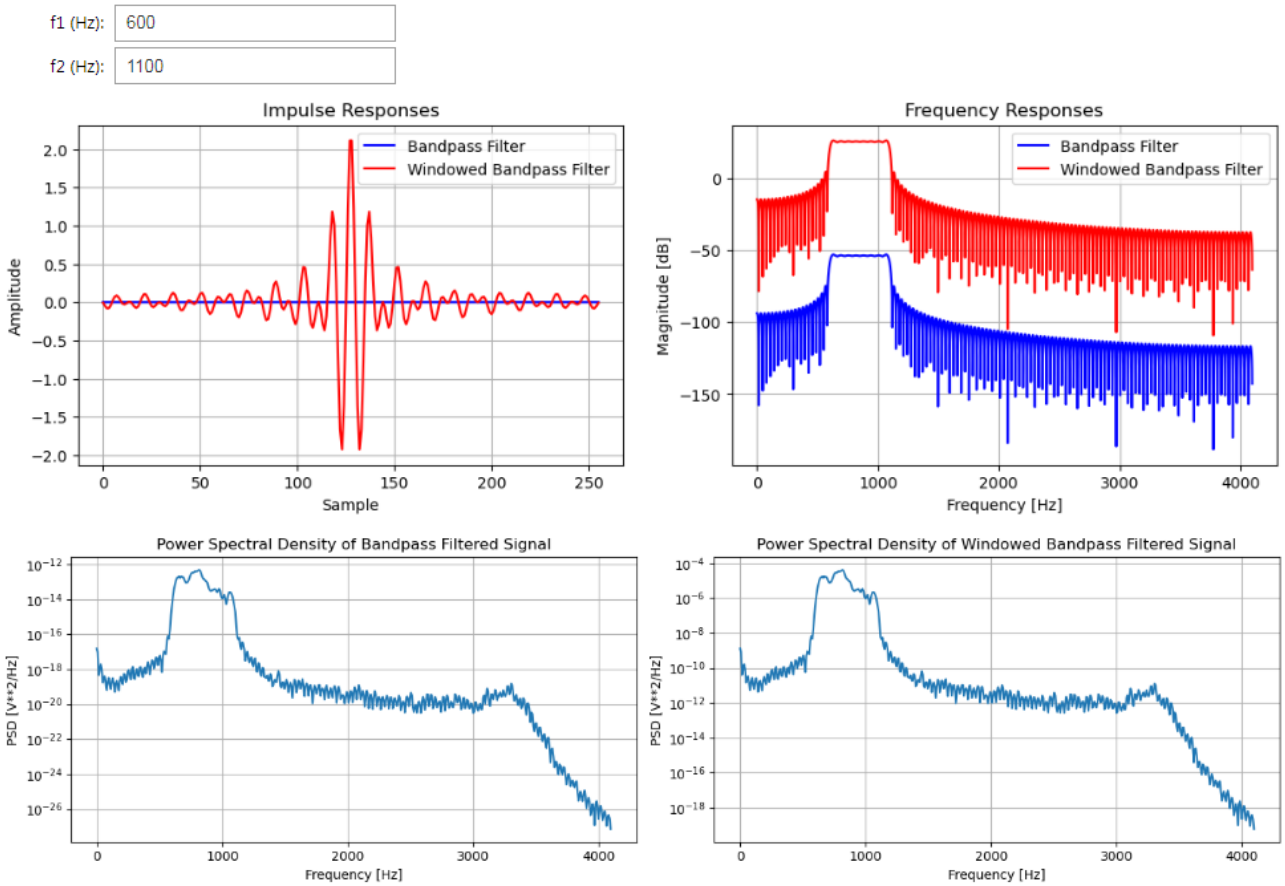


Σχήμα 9: Φασματική Πυκνότητα Ισχύος (Power Spectral Density)

3.3.1.10 Ανάλυση Απόκρισης Ζωνοπερατών Φίλτρων

Εφαρμογή ζωνοπερατών φίλτρων και οπτική απεικόνιση τόσο της απόκρισης συχνότητας αυτών, όσο και της φασματικής πυκνότητας του φιλτραρισμένου σήματος.

Κώδικας – Chapter 1 - Ανάλυση Απόκρισης Bandpass Φίλτρων

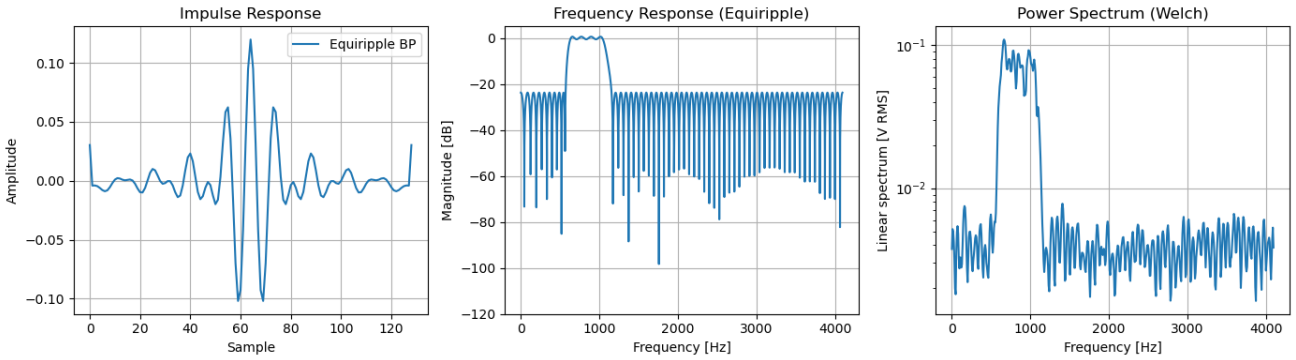


Σχήμα 10: Ανάλυση Απόκρισης Ζωνοπερατών Φίλτρων

3.3.1.11 Εφαρμογή ζωνοπερατού Φίλτρου Parks-McClellan

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

Κώδικας – Chapter 1 - Εφαρμογή Parks-McClellan Φίλτρου

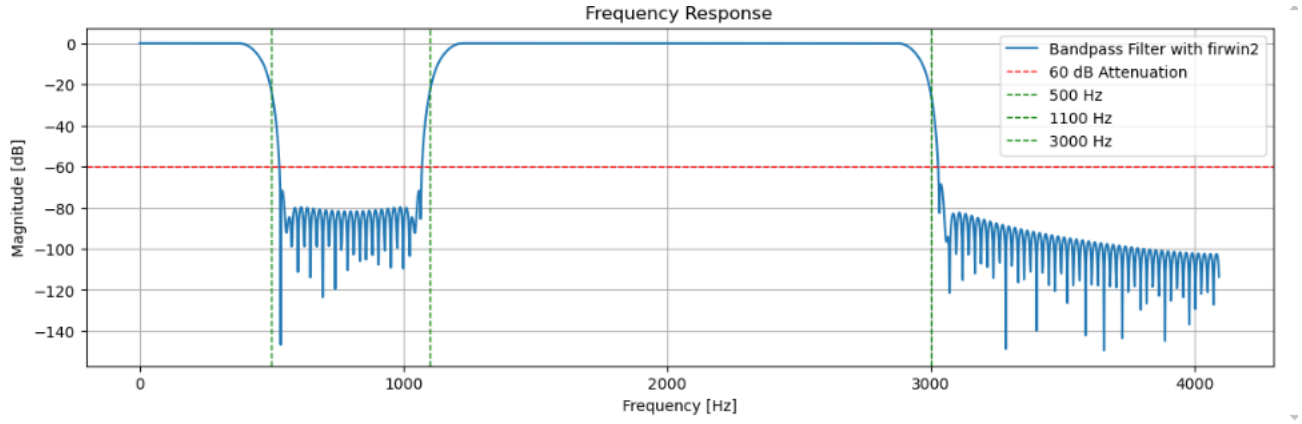


Σχήμα 11: Εφαρμογή ζωνοπερατού Φίλτρου Parks-McClellan

3.3.1.12 Εφαρμογή Φίλτρου με δύο ζώνες διέλευσης

Σχεδιασμός και υλοποίηση φίλτρου με δύο ζώνες διέλευσης, με απεικόνιση της φασματικής του απόκρισης. Ο σχεδιασμός του φίλτρου πραγματοποιήθηκε με τη μέθοδο FIR firwin2 (frequency-sampling) και παράθυρο amming, τάξης 350, για Fs = 8192 z.

Κώδικας – Chapter 1 - Εφαρμογή Φίλτρου με 2 passbands



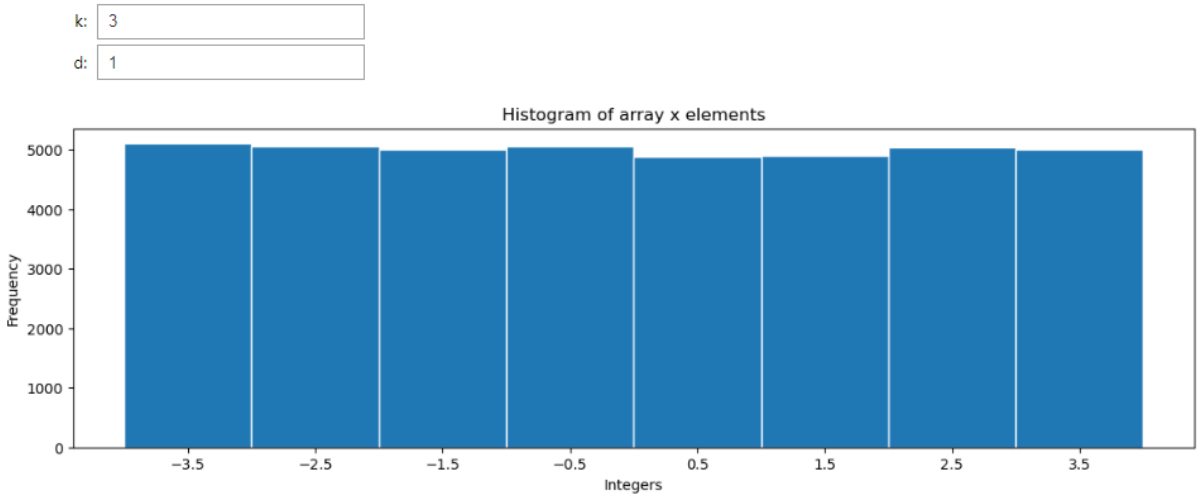
Σχήμα 12: Εφαρμογή Φίλτρου με δύο ζώνες διέλευσης

3.3.2 Βέλτιστη ψηφιακή αναγνώριση – Προσαρμοσμένα φίλτρα

3.3.2.1 Δημιουργία τυχαίων συμβόλων διαμόρφωσης ASK

Δημιουργία τυχαίων συμβόλων L-ASK ($L=2^k$) απόστασης d και παρουσίαση του σχετικού ιστογράμματος.

Κώδικας – Chapter 3 - Δημιουργία τυχαίων σημάτων

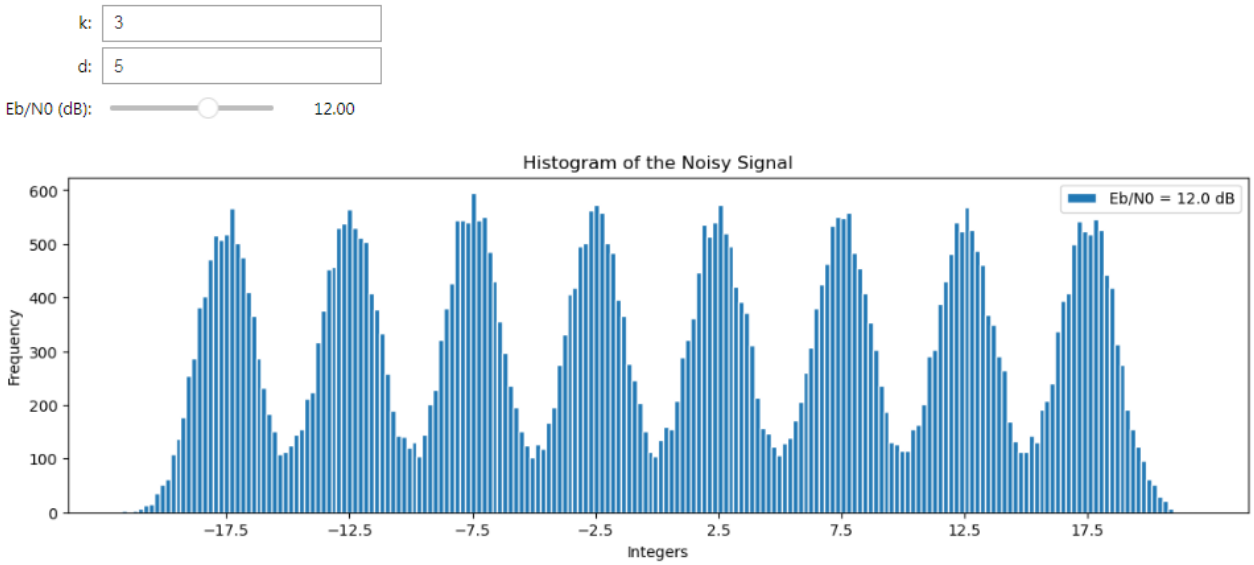


Σχήμα 13: Δημιουργία τυχαίων συμβόλων διαμόρφωσης ASK

3.3.2.2 Προσθήκη θορύβου

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

Κώδικας – Chapter 3 - Εφαρμογή θορύβου

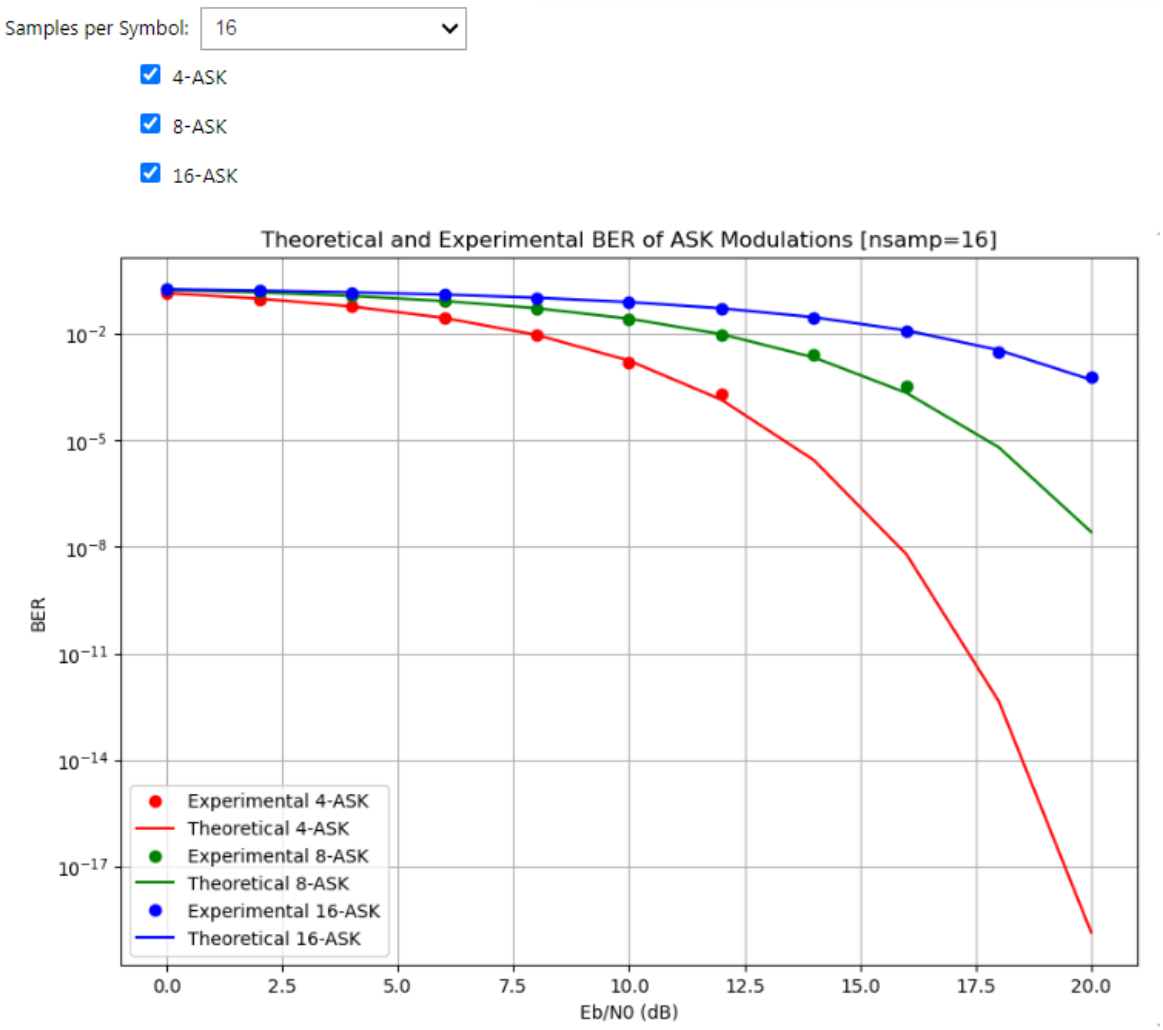


Σχήμα 14: Προσθήκη θορύβου

3.3.2.3 Οπτικοποίηση ASK Bit Error Rate (BER)

Η διαδραστική δυνατότητα σύγκρισης διαμορφώσεων L-ASK με επιλογές για πειραματική και θεωρητική τιμή του BER. Οι χρήστες μπορούν να επιλέξουν μεταξύ διαφορετικών ASK διαμορφώσεων και matched filter με αντιστροφή ή όχι για να συγκρίνονται αποτελέσματα.

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

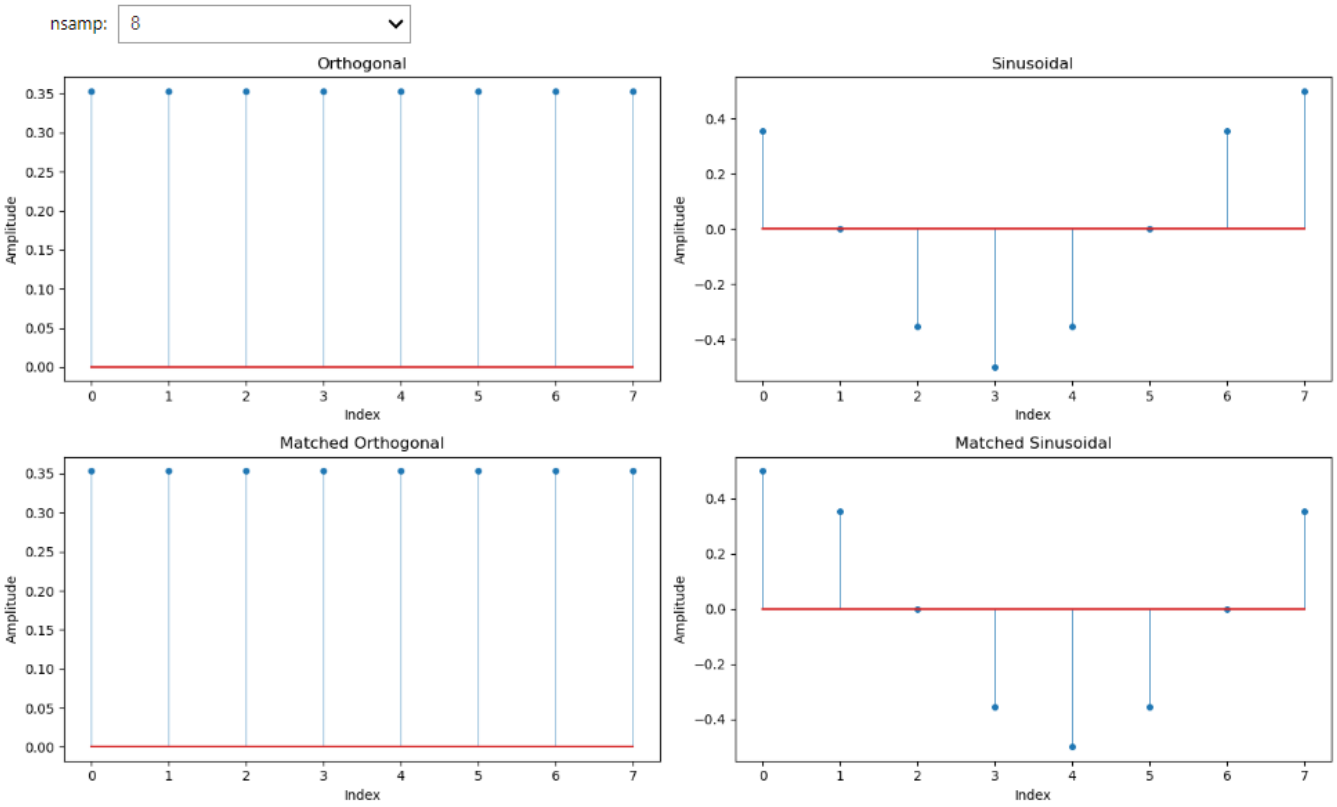


Σχήμα 15: Οπτικοποίηση ASK Bit Error Rate (BER)

3.3.2.4 Ανάλυση με φίλτρα

Δίνεται η δυνατότητα οπτικοποίησης της κρουστικής απόκρισης φίλτρων ορθογωνικών ή ημιτονοειδών με διαφορετικό μήκος (nsamp) ώστε να μπορεί κάποιος να παρατηρήσει την ασυμμετρία του φίλτρου σε χαμηλές nsamp τιμές.

Κώδικας – Chapter 3 – Ανάλυση με φίλτρα



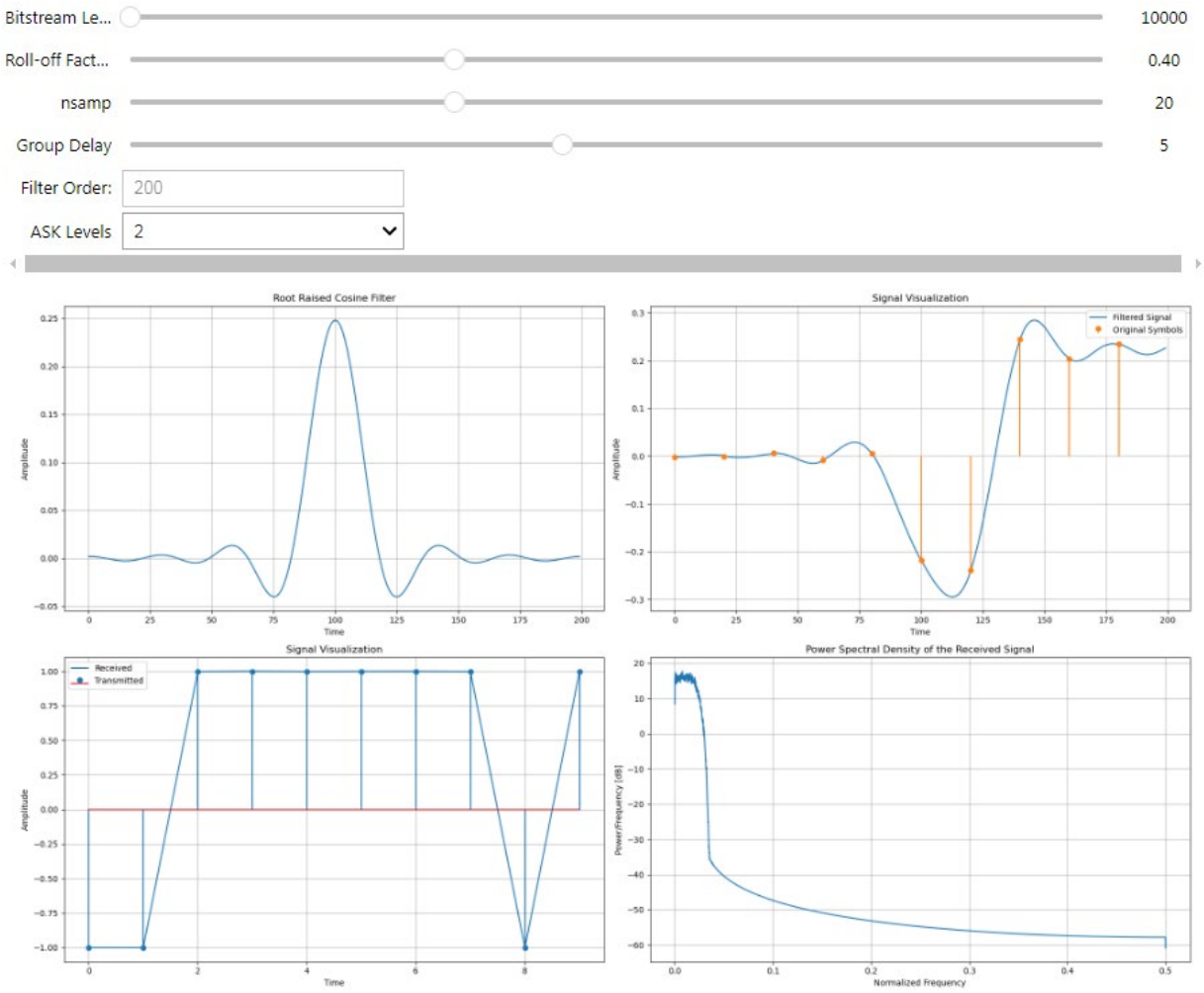
Σχήμα 16: Ανάλυση με φίλτρα

3.3.3 Φασματικά χαρακτηριστικά – Φίλτρα Nyquist

3.3.3.1 Ρυθμιζόμενες παράμετροι

Εισαγωγή διαδραστικού τρόπου ρύθμισης μήκους ψηφιοσειράς (bit stream), και παραμέτρων του Nyquist φίλτρου όπως είναι ο συντελεστής roll-off, τα δείγματα ανά σύμβολο (nsamp), το group delay, που καθορίζουν την τάξη του φίλτρου.

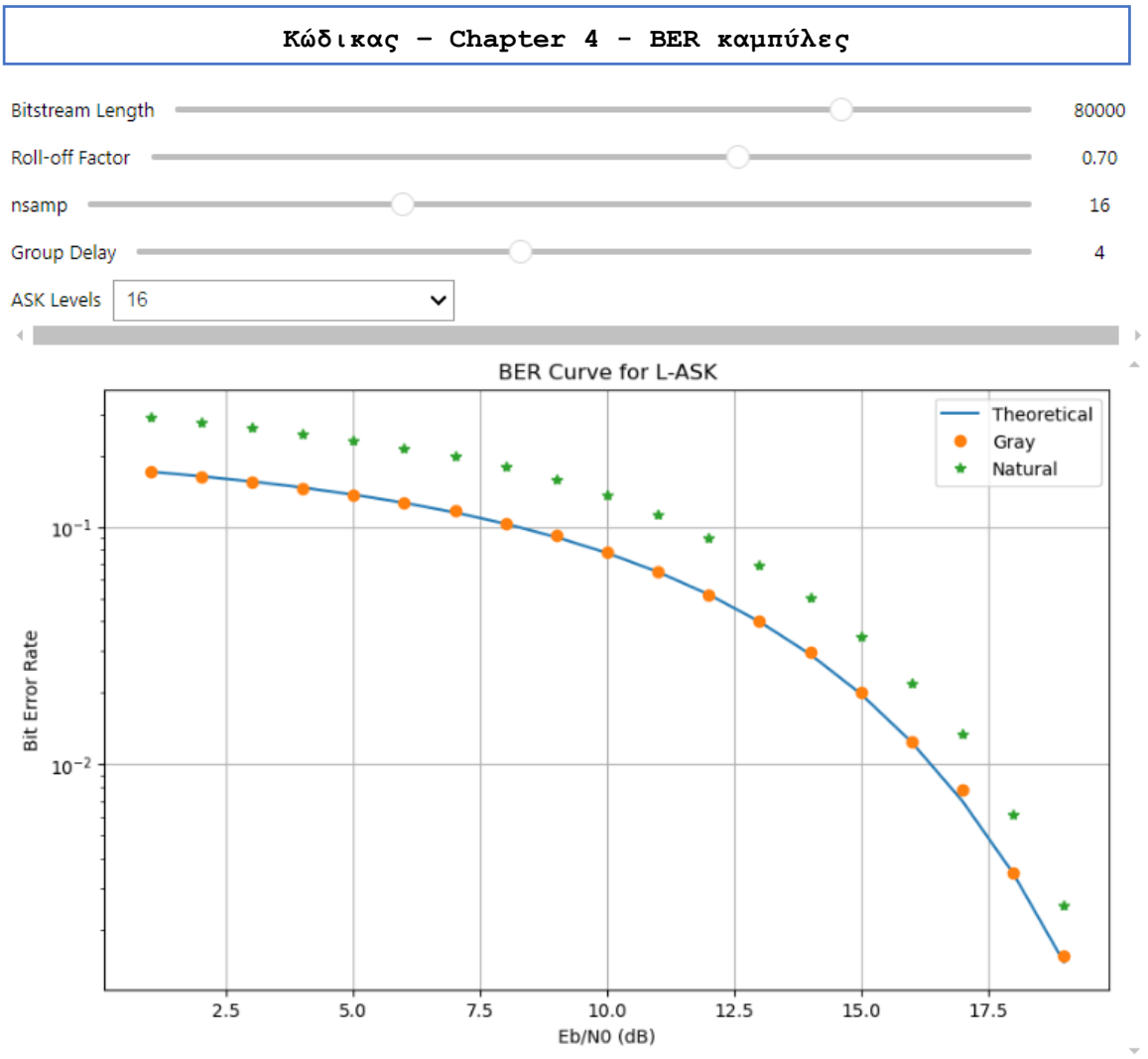
Κώδικας – Chapter 4 - Ρυθμιζόμενες παράμετροι



Σχήμα 17: Ρυθμιζόμενες παράμετροι

3.3.3.2 Καμπύλες BER

Με δυνατότητα επιλογής της τάξης διαμόρφωσης L-ASK, εμφάνισης των θεωρητικών και πειραματικών καμπυλών σε σχέση με το Eb/No και σύγκριση μεταξύ κωδικοποίησης Gray και άλλης.



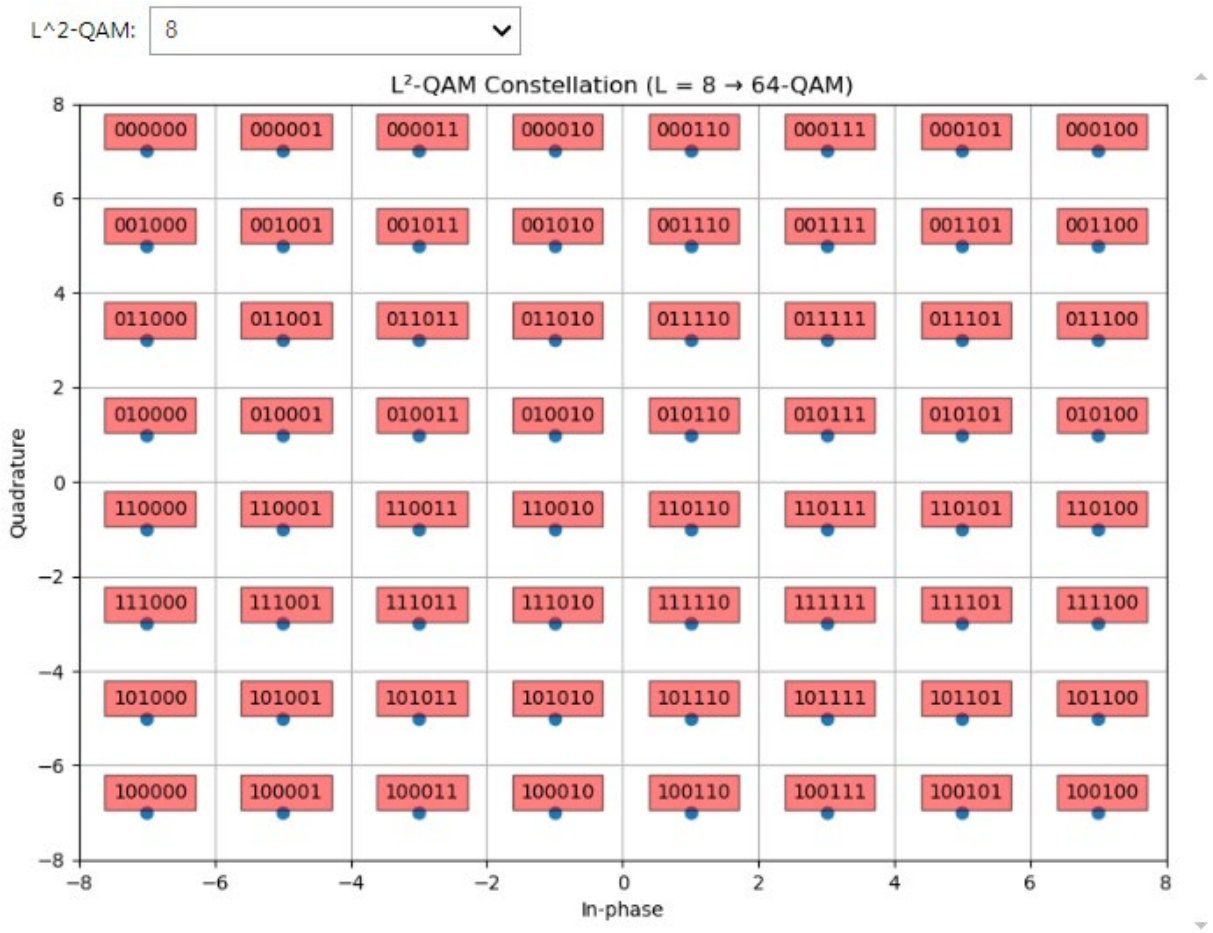
Σχήμα 18: Καμπύλες BER

3.3.4 Ψηφιακή Διαμόρφωση QAM και PSK

3.3.4.1 Σηματικοί αστερισμοί QAM και PSK

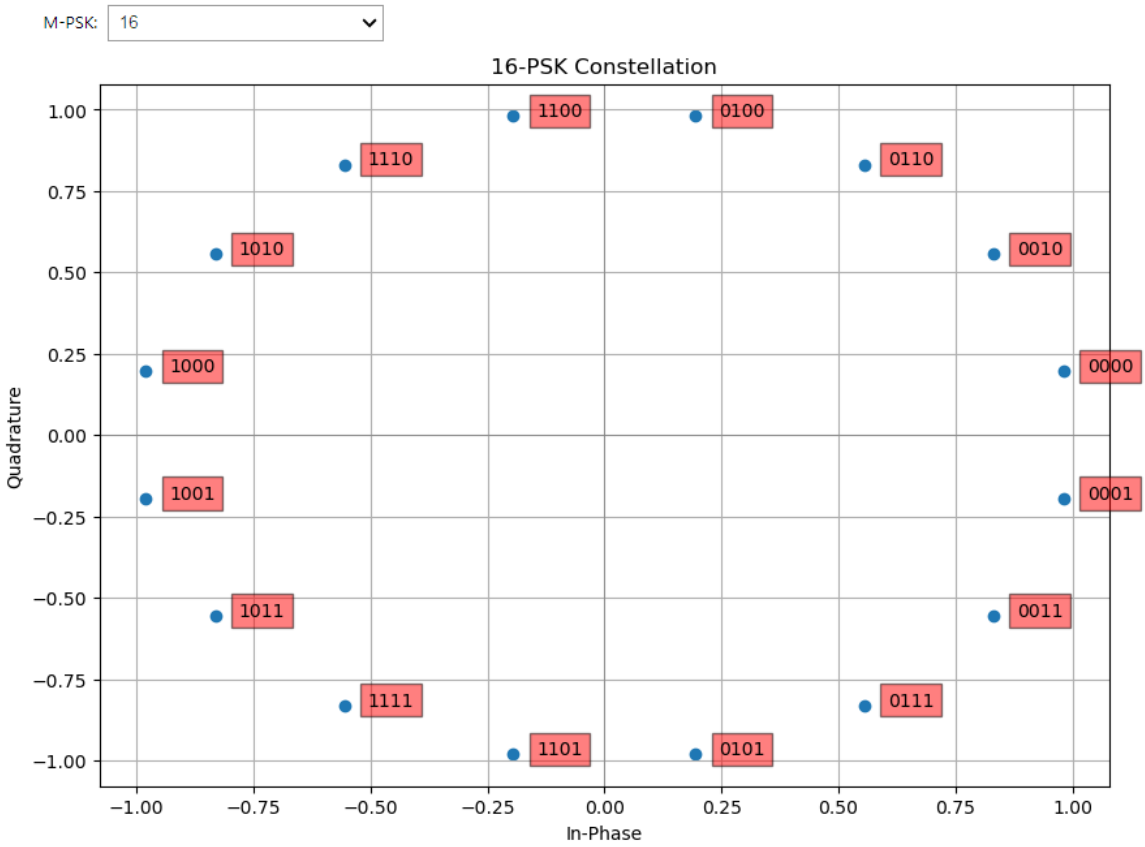
Διαδραστικά διαγράμματα αστερισμών για M-QAM και M-PSK. Οι χρήστες μπορούν να επιλέξουν το επίπεδο διαμόρφωσης και να δουν τις αντίστοιχες θέσεις των συμβόλων στον αστερισμό.

Κώδικας – Chapter 5 - QAM Constellation



Σχήμα 19: Αστερισμός QAM

Κώδικας – Chapter 5 - PSK Constellation

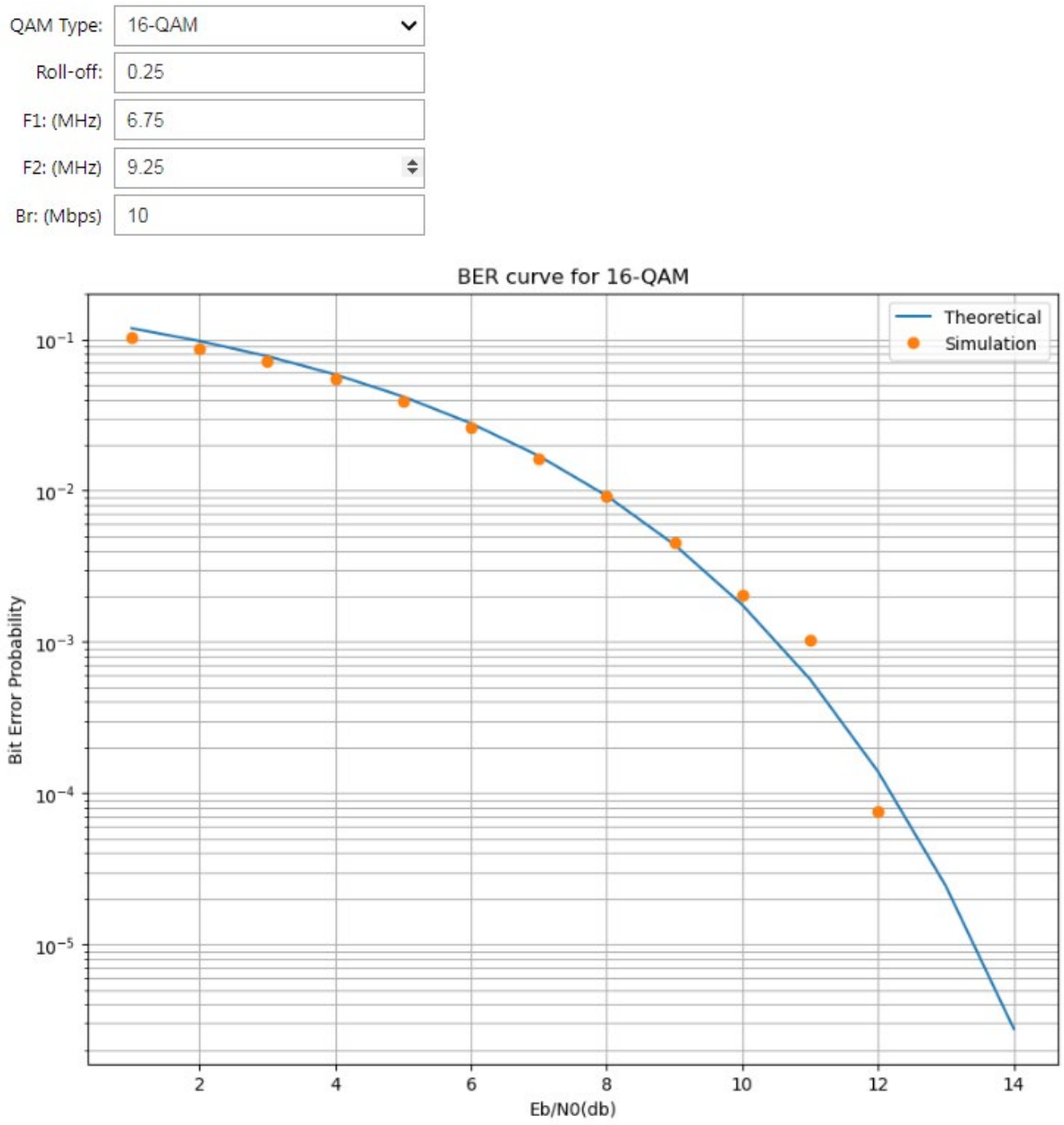


Σχήμα 20: Αστερισμός PSK

3.3.4.2 QAM και PSK Καμπύλες BER

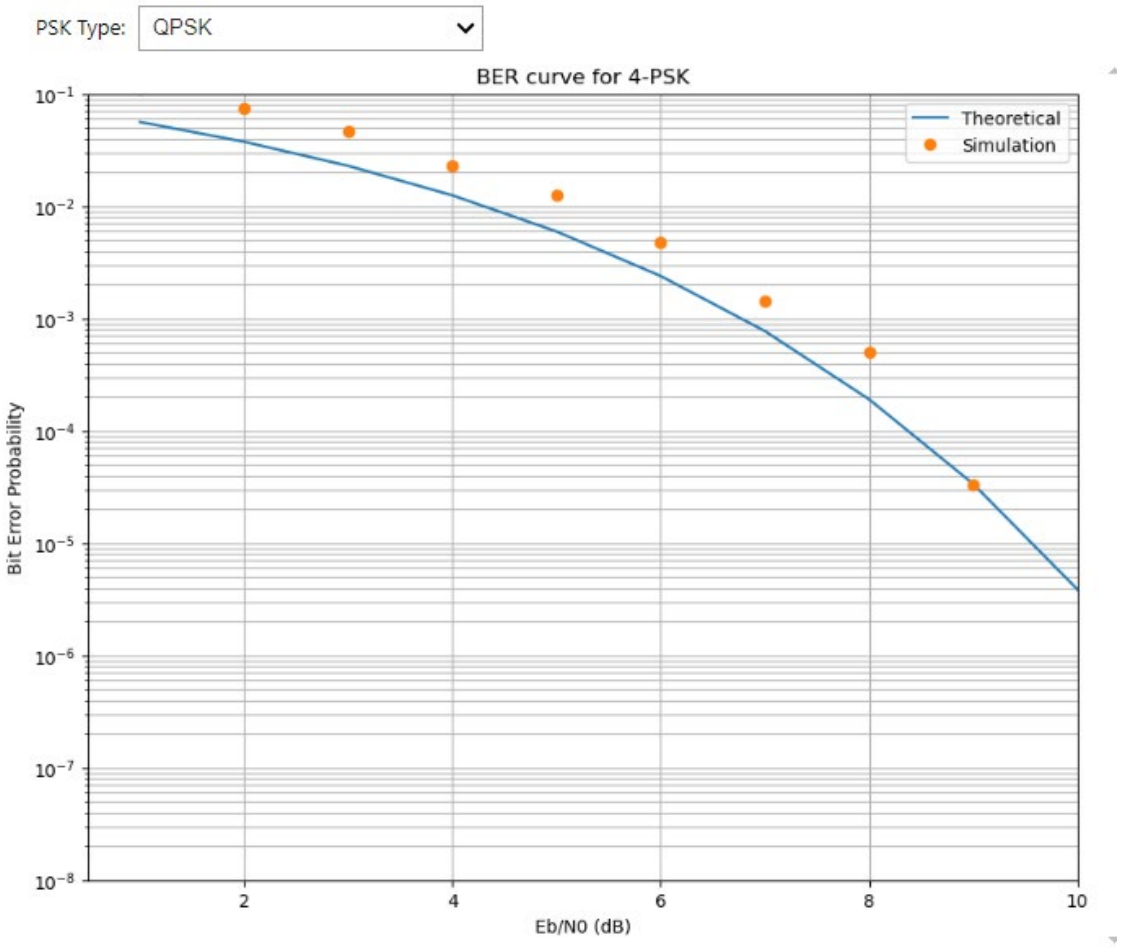
Δημιουργία γραφημάτων για τις καμπύλες BER τόσο για QAM όσο και για PSK διαμορφώσεις, όπου οι χρήστες μπορούν να συγκρίνουν τη θεωρητική και πειραματική καμπύλη και να ρυθμίσουν παραμέτρους όπως το roll-off του φίλτρου Nyquist, τις συχνότητες f1 και f2 του ζωνοπερατού δίαυλου, και τον επιθυμητό ρυθμό μετάδοσης για την ανάλυση της πιθανότητας σφάλματος.

Κώδικας – Chapter 5 - QAM BER



Σχήμα 21: QAM BER

Κώδικας – Chapter 5 - PSK BER

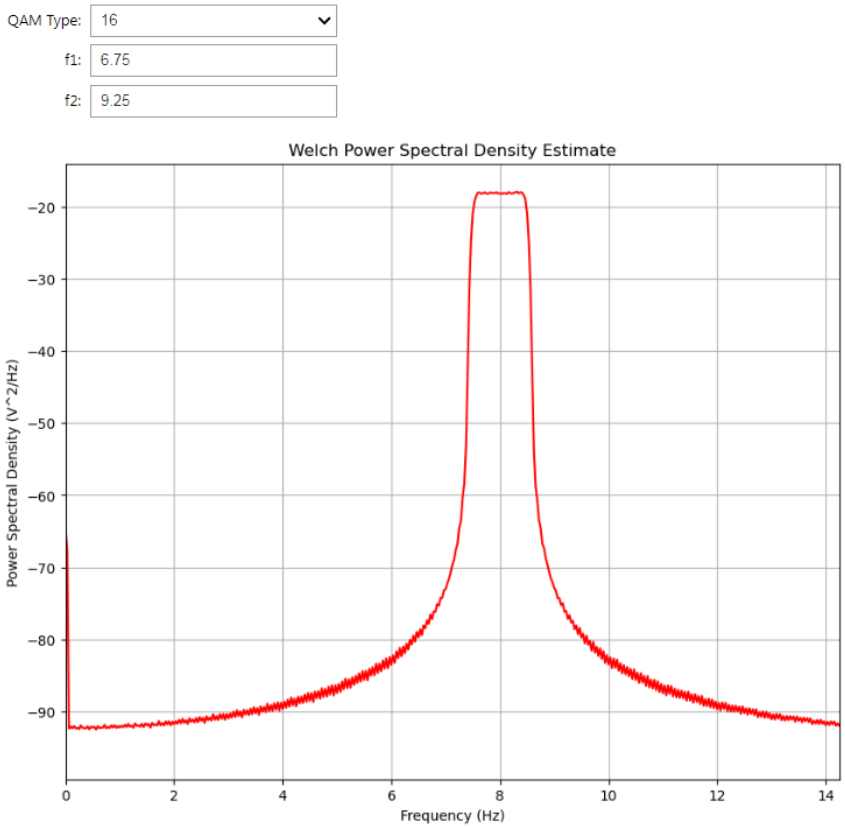


Σχήμα 22: PSK BER

3.3.4.3 Διαδραστική Φασματική Πυκνότητα Ισχύος

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

Κώδικας – Chapter 5 – Διαδραστική Φασματική Πυκνότητα Ισχύος

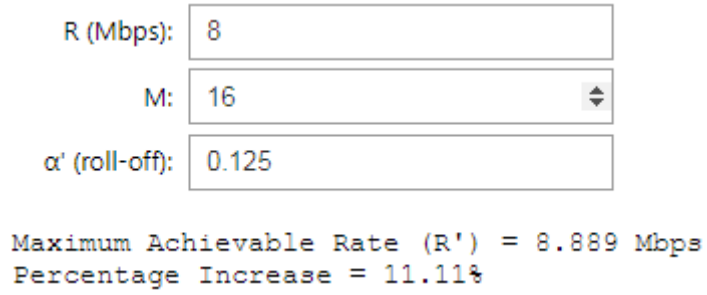


Σχήμα 23: Διαδραστική Φασματική Πυκνότητα Ισχύος

3.3.4.4 Διαδραστικός Υπολογισμός Παραμέτρων

Δυνατότητα υπολογισμού νέου ρυθμού μετάδοσης και αντίστοιχης ποσοστιαίας μεταβολής, μεταβάλλοντας είτε την τάξη της διαμόρφωσης είτε τον συντελεστή roll-off του φίλτρου.

Κώδικας – Chapter 5 – Διαδραστικός Υπολογισμός Παραμέτρων



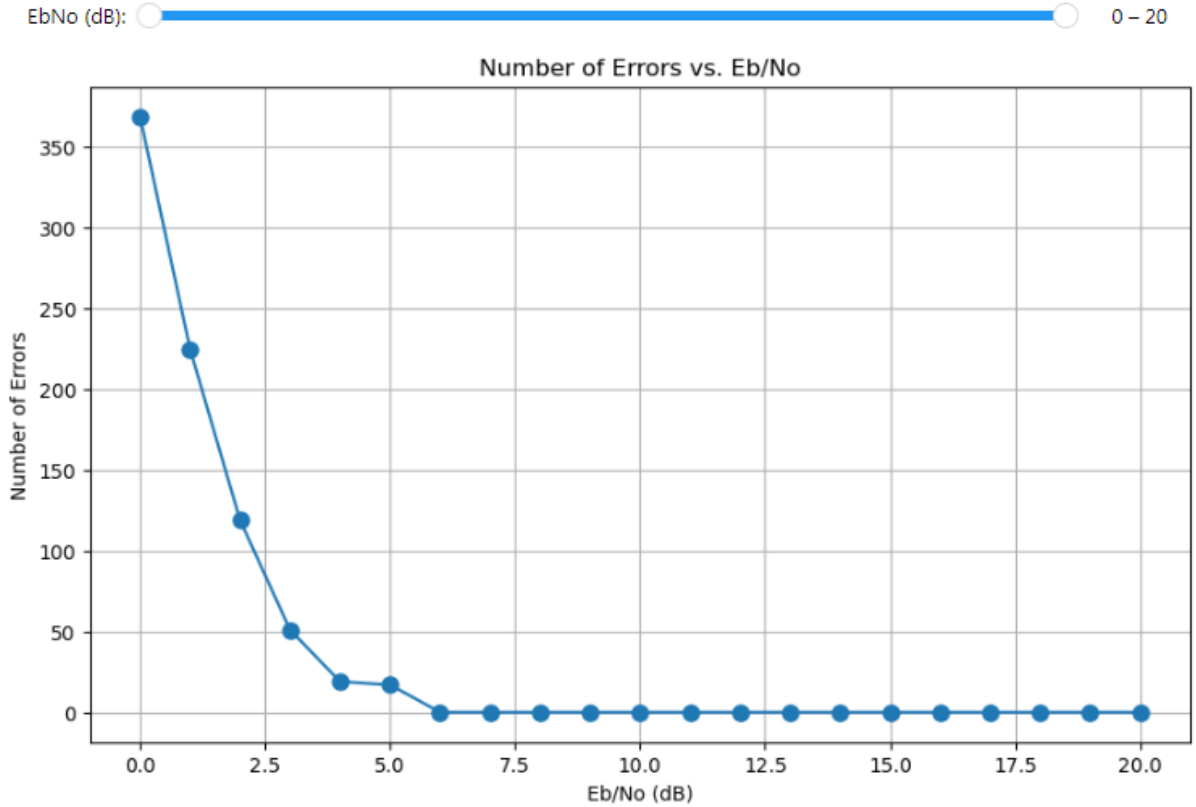
Σχήμα 24: Διαδραστικός Υπολογισμός Παραμέτρων

3.3.5 Ψηφιακή Διαμόρφωση FSK και MSK

3.3.5.1 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

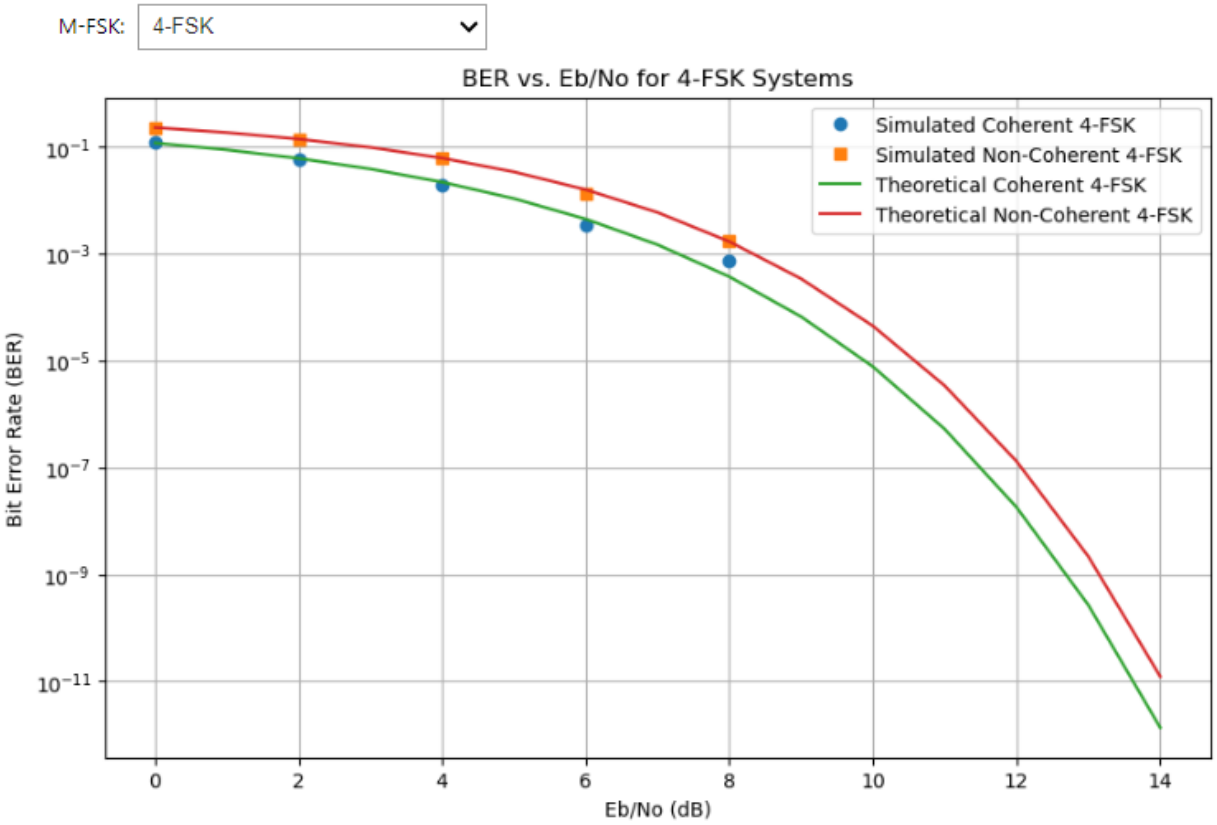


Σχήμα 25: Αριθμός σφαλμάτων για Eb/No

3.3.5.2 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 τη θεωρητική και πειραματική απόδοση.

Κώδικας – Chapter 6 - FSK Bit Error Rate

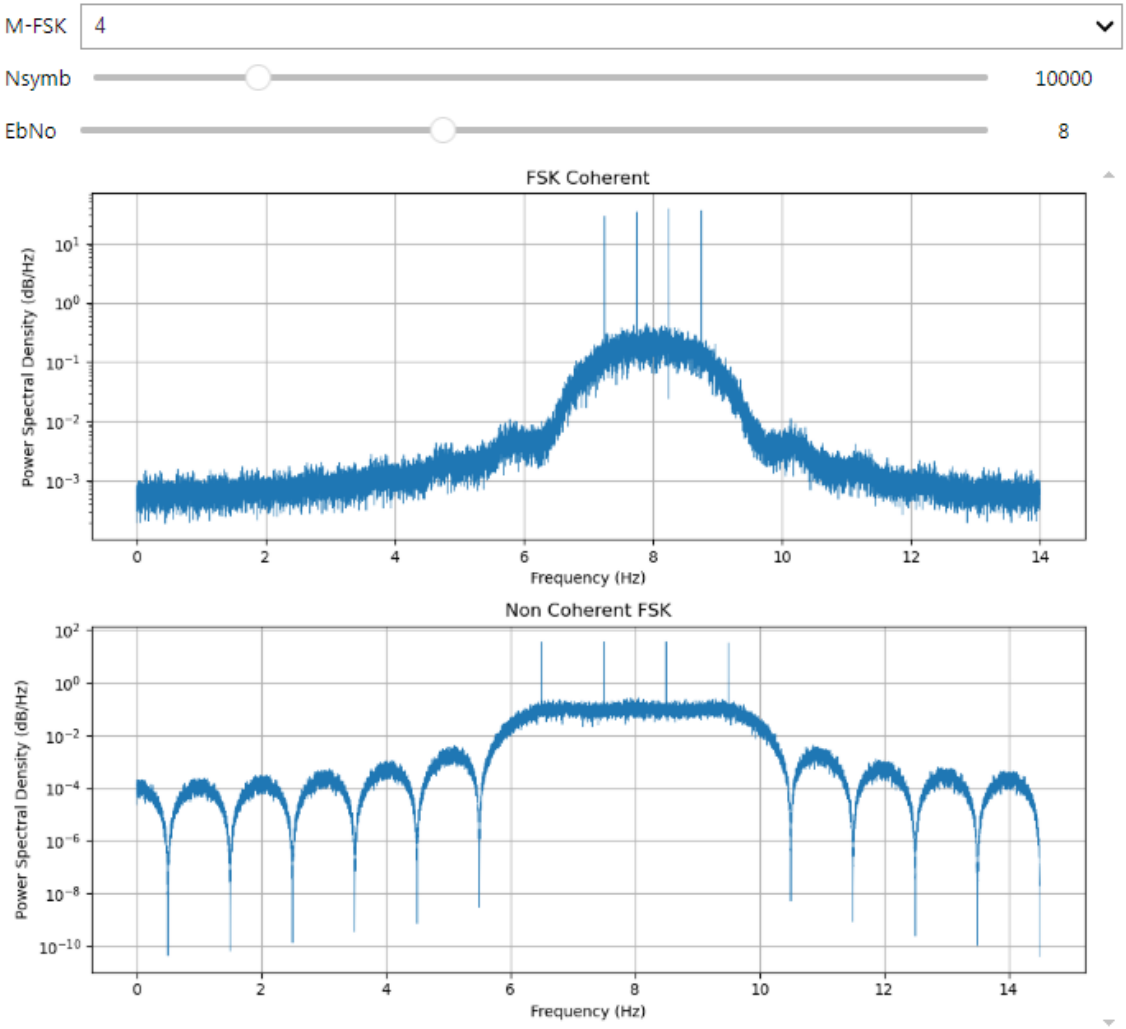


Σχήμα 26: FSK Bit Error Rate

3.3.5.3 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 τον αριθμό συμβόλων και το Eb/No.

Κώδικας – Chapter 6 - Analysis coherent και non coherent FSK

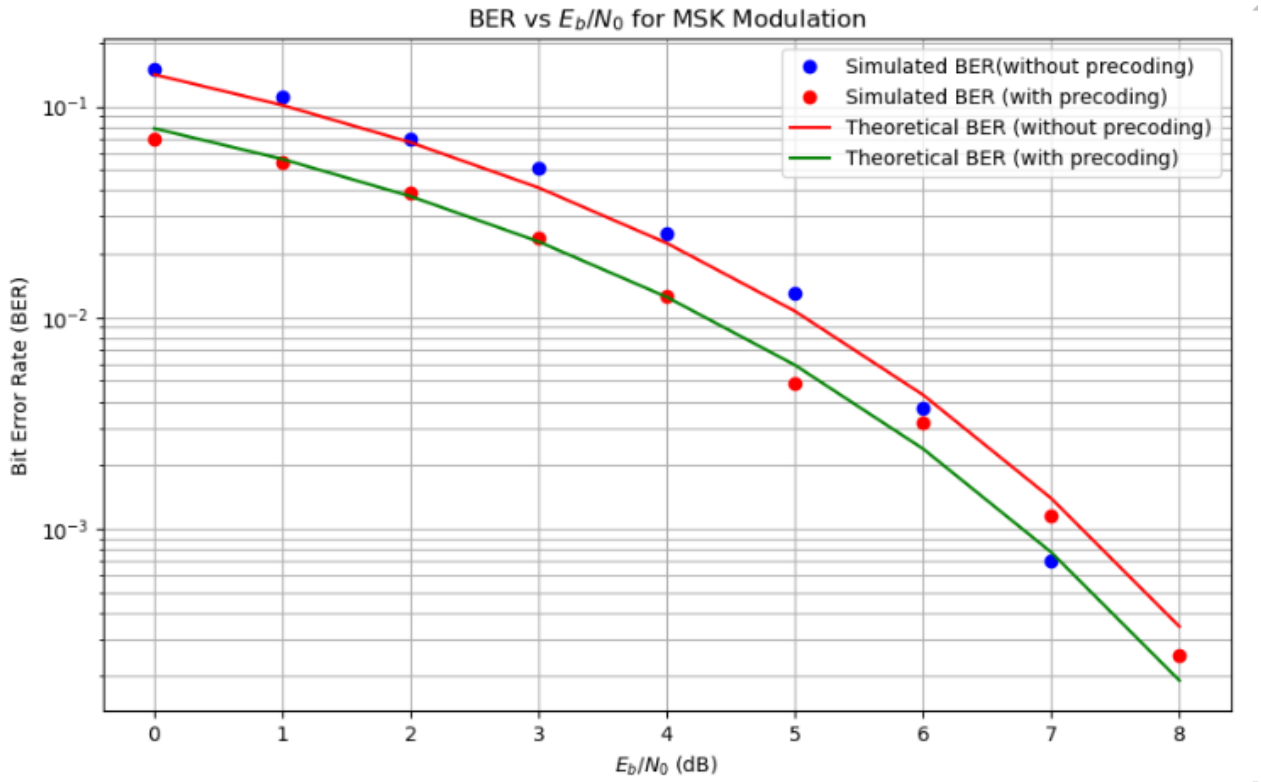


Σχήμα 27: Ανάλυση Φάσματος coherent και non coherent FSK

3.3.5.4 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



Σχήμα 28: MSK Bit Error Rate

3.4 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 [46]. Με αυτόν τον τρόπο, επιτυγχάνεται μια πιο καθαρή και άμεση παρουσίαση των αποτελεσμάτων, χωρίς περιττές πληροφορίες.

Κώδικας – Custom JavaScript

3.5 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 Gitub Pages για την ανάπτυξη και φιλοξενία της διαδικτυακής εφαρμογής.

  Οδηγίες - Βήμα 8

- Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για τη δημοσίευση της ιστοσελίδας στο Gitub Page που δημιουργήσαμε.

  Οδηγίες - Βήμα 9

Κεφάλαιο 4

Σύνοψη, συμπεράσματα και μελλοντικές επεκτάσεις

4.1 Εισαγωγή

Στο παρόν κεφάλαιο συνοψίζονται τα σημαντικότερα ευρήματα και συμπεράσματα που προέκυψαν από την υλοποίηση και αξιολόγηση των εφαρμογών προσομοίωσης που πραγματοποιήθηκαν. Αρχικά, παρουσιάζονται συνοπτικά οι στόχοι της εργασίας και τα βασικά στάδια της μεθοδολογικής προσέγγισης. Στη συνέχεια, προτείνονται πιθανές επεκτάσεις της παρούσας εργασίας για περαιτέρω έρευνα και ανάπτυξη. Επιπρόσθετα, αναλύονται συγκριτικά τα προσομοιωμένα αποτελέσματα με τα αντίστοιχα θεωρητικά αναμενόμενα, με έμφαση στις αποκλίσεις και στους χρόνους εκτέλεσης. Τέλος, συζητούνται οι βασικές προκλήσεις και δυσκολίες που αντιμετωπίστηκαν κατά την ανάπτυξη των προσομοιώσεων, καθώς και η επίδρασή τους στην ακρίβεια και στην απόδοση των τελικών αποτελεσμάτων.

4.2 Στόχοι της Εργασίας

Οι στόχοι της παρούσας εργασίας επικεντρώθηκαν στην αναζήτηση, στην αξιολόγηση, στην επιλογή και στη 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 ήταν 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 επικεντρώθηκε σε εργαλεία ανοικτού 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, αποτέλεσε κεντρικό εργαλείο στην υλοποίηση των στόχων αυτών. Τέλος, απαραίτητο στοιχείο αποτέλεσε 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.

4.3 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 [45].

1. **Αναζήτηση, δοκιμή και τελική επιλογή των κατάλληλων εργαλείων για την ολοκλήρωση του project.**

2. **Δημιουργία αποθετηρίου στο Gitub για τη διαχείριση των αρχείων του έργου.**

3. **Κατασκευή βασικών κομματιών κώδικα python γύρω από τη 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 εργαλείου 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 Gitub Pages για την ανάπτυξη και φιλοξενία της διαδικτυακής εφαρμογής.**

9. **Εκτέλεση των κατάλληλων εντολών μέσω του Jupyter Book για τη δημοσίευση της ιστοσελίδας στο Gitub 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 υπάρχοντα 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 τη βοήθεια των κομματιών κώδικα από το βήμα 3)

15. Προσθήκη διαδραστικών λειτουργιών στον κώδικα Python μέσω των κατάλληλων βιβλιοθηκών. (Με τη 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 τη δημοσίευση της ιστοσελίδας στο Gitub Page που δημιουργήσαμε.

4.4 Μελλοντικές Επεκτάσεις

4.4.1 Επέκταση σε Εργαστηριακές Ασκήσεις

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

4.4.2 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 του scikit-learn για προχωρημένες αναλύσεις.
3. **Ανάπτυξη Mobile Εφαρμογής:** Χρήση πλαισίων όπως του Kivy για πρόσβαση από κινητές συσκευές.
4. **Διεθνοποίηση (i18n):** Υποστήριξη πολλαπλών γλωσσών για ευρύτερο κοινό.
5. **Αυτοματισμός με CI/CD:** Ενσωμάτωση εργαλείων όπως του Gitub 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 του Google Colab για συνεργατική εργασία.
10. **Προηγμένες Οπτικοποιήσεις 3D:** Χρήση βιβλιοθηκών όπως του VisPy για τριδιάστατες αναπαραστάσεις.
11. **Χρήση Docker Containers:** Για ευκολότερη ανάπτυξη και μεταφορά της εφαρμογής.

4.5 Συμπεράσματα Προσομοιώσεων

Γενικά, διαπιστώνεται ότι οι υλοποιήσεις προσομοιώσεων σε MATLAB παρουσιάζουν καλύτερους χρόνους εκτέλεσης σε σύγκριση με τις αντίστοιχες υλοποιήσεις σε Python, ακόμη και όταν γίνεται χρήση εξειδικευμένων βιβλιοθηκών. Η διαφορά αυτή μπορεί να αποδοθεί στο γεγονός ότι το MATLAB είναι ειδικά σχεδιασμένο για αριθμητικούς και μαθηματικούς υπολογισμούς, παρέχοντας ιδιαίτερα βελτιστοποιημένες λειτουργίες για συστήματα ψηφιακών επικοινωνιών, όπως εκείνα που εξετάστηκαν στην παρούσα εργασία. Ειδικότερα, οι ενσωματωμένες βιβλιοθήκες του MATLAB, όπως το Communications Toolbox, αξιοποιούν αποτελεσματικά τις δυνατότητες πολυνηματικής επεξεργασίας και υποστήριξης GPU.

Αντίθετα, η Python, αν και προσφέρει ισχυρές και ευέλικτες βιβλιοθήκες ανοικτού κώδικα, όπως η NumPy και η SciPy, συχνά απαιτεί πρόσθετες τεχνικές βελτιστοποίησης για να προσεγγίσει τις επιδόσεις του MATLAB. Επιπλέον, η υλοποίηση σε Python περιλαμβάνει πρόσθετες διαδραστικές δυνατότητες, γεγονός που καθιστά τη σύγκριση των δύο υλοποιήσεων εν μέρει μη ισοδύναμη.

Συνολικά, για την πλειονότητα των εφαρμογών που εξετάστηκαν, οι διαφορές στους χρόνους εκτέλεσης ήταν αμελητέες, δεδομένου ότι ήταν αρκετά μικροί και στις δύο περιπτώσεις. Ωστόσο, ειδικά στα γραφήματα του ποσοστού σφάλματος δυαδικών ψηφίων (Bit Error Rate - BER), παρατηρήθηκε σημαντική επιβράδυνση της Python σε σχέση με το MATLAB. Η διαφορά αυτή είναι αποτέλεσμα τόσο της εξειδίκευσης του MATLAB σε αριθμητικούς υπολογισμούς όσο και της αποτελεσματικότερης αξιοποίησης των υπολογιστικών πόρων που προσφέρει.

Πηγές

[1] Ν. Μήτρου, «*Ψηφιακές Επικοινωνίες* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις» 2015 . [Ηλεκτρονικό]. Available: https://dx.doi.org/10.57713/kallipos-454.

[2] Ν. Μήτρου, «*Η Ψηφιακή Επεξεργασία Σήματος στις Τηλεπικοινωνίες* [Κεφάλαιο]. Στο Μήτρου, Ν. 2015. *Ψηφιακές Επικοινωνίες* [Προπτυχιακό εγχειρίδιο]». Available: https://repository.kallipos.gr/handle/11419/6045.

[3] Ν. Μήτρου, «*Βέλτιστη ψηφιακή αναγνώριση – Προσαρμοσμένα φίλτρα [Κεφάλαιο].* Στο Μήτρου, Ν. 2015. *Ψηφιακές Επικοινωνίες* [Προπτυχιακό εγχειρίδιο]». Available: https://repository.kallipos.gr/handle/11419/6047.

[4] Ν. Μήτρου, «*Φασματικά χαρακτηριστικά ψηφιακών κυματομορφών – Μορφοποίηση με φίλτρα Nyquist* [Κεφάλαιο]. Στο Μήτρου, Ν. 2015. *Ψηφιακές Επικοινωνίες* [Προπτυχιακό εγχειρίδιο]» Available: https://repository.kallipos.gr/handle/11419/6048.

[5] Ν. Μήτρου, «*Ψηφιακή Διαμόρφωση QAM και PSK [Κεφάλαιο].* Στο Μήτρου, Ν. 2015. *Ψηφιακές Επικοινωνίες* [Προπτυχιακό εγχειρίδιο]» Available: https://repository.kallipos.gr/handle/11419/6049.

[6] Ν. Μήτρου, «*Ψηφιακή διαμόρφωση FSK και MSK [Κεφάλαιο].* Στο Μήτρου, Ν. 2015. *Ψηφιακές Επικοινωνίες* [Προπτυχιακό εγχειρίδιο]». Available: https://repository.kallipos.gr/handle/11419/6050.

[7] «Gitub - Documentation,» [Ηλεκτρονικό]. Available: https://docs.github.com/en.

[8] «Jupyter Book - Documentation,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/intro.html.

[9] «Voila - Documentation,» [Ηλεκτρονικό]. Available: https://voila.readthedocs.io/en/stable/.

[10] «MkDocs - Documentation,» [Ηλεκτρονικό]. Available: (https://www.mkdocs.org/).

[11] «Dash - Documentation,» [Ηλεκτρονικό]. Available: https://dash.plotly.com/.

[12] «Google Colab - Documentation,» [Ηλεκτρονικό]. Available: https://colab.research.google.com/notebooks/intro.ipynb.

[13] «Jupyter Book - Notebook Interface,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/interactive/launchbuttons.html#control-the-notebook-interface-that-opens.

[14] «Jupyter Book - Binder,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/interactive/launchbuttons.html#add-a-launch-on-binder-button.

[15] «Jupyter Book - Jupyterub,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/interactive/launchbuttons.html#add-a-launch-on-jupyterhub-button.

[16] «Jupyter Book - Thebe,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/interactive/thebe.html.

[17] «Jupyter Book – Google Colab,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/interactive/launchbuttons.html#add-a-launch-on-google-colab-button.

[18] «NumPy - Documentation,» [Ηλεκτρονικό]. Available: https://numpy.org/doc/.

[19] «SciPy - Documentation,» [Ηλεκτρονικό]. Available: https://docs.scipy.org/doc/scipy/reference/.

[20] «scipy.signal - Documentation,» [Ηλεκτρονικό]. Available: https://docs.scipy.org/doc/scipy/reference/signal.html.

[21] «CommPy - Documentation,» [Ηλεκτρονικό]. Available: https://github.com/veeresht/CommPy.

[22] «Math Module - Documentation,» [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/math.html.

[23] «Matplotlib - Documentation,» [Ηλεκτρονικό]. Available: https://matplotlib.org/stable/contents.html.

[24] «SimPy - Documentation,» [Ηλεκτρονικό]. Available: https://simpy.readthedocs.io/en/latest/.

[25] «PyWavelets - Documentation,» [Ηλεκτρονικό]. Available: https://pywavelets.readthedocs.io/en/latest/.

[26] «scikit-dsp-comm - Documentation,» [Ηλεκτρονικό]. Available: https://scikit-dsp-comm.readthedocs.io/en/latest/.

[27] «PySDR Gitub,» [Ηλεκτρονικό]. Available: https://github.com/ve3wwg/pySDR.

[28] «Seaborn - Documentation,» [Ηλεκτρονικό]. Available: https://seaborn.pydata.org/.

[29] «Plotly - Documentation,» [Ηλεκτρονικό]. Available: https://plotly.com/python/.

[30] «Bokeh - Documentation,» [Ηλεκτρονικό]. Available: https://docs.bokeh.org/en/latest/.

[31] «PyPlot - Documentation,» [Ηλεκτρονικό]. Available: https://matplotlib.org/stable/api/pyplot_summary.html.

[32] «ggplot - Documentation,» [Ηλεκτρονικό]. Available: http://ggplot.yhathq.com/.

[33] «Altair - Documentation,» [Ηλεκτρονικό]. Available: https://altair-viz.github.io/.

[34] «Pandas Visualization - Documentation,» [Ηλεκτρονικό]. Available: https://pandas.pydata.org/docs/user_guide/visualization.html.

[35] «oloviews - Documentation,» [Ηλεκτρονικό]. Available: https://holoviews.org/.

[36] «VisPy - Documentation,» [Ηλεκτρονικό]. Available: http://vispy.org/.

[37] «IPython - Documentation,» [Ηλεκτρονικό]. Available: https://ipython.org/documentation.html.

[38] «ipywidgets - Documentation,» [Ηλεκτρονικό]. Available: https://ipywidgets.readthedocs.io/en/stable/.

[39] «Plotly Dash - Documentation,» [Ηλεκτρονικό]. Available: https://dash.plotly.com/.

[40] «Streamlit - Documentation,» [Ηλεκτρονικό]. Available: https://docs.streamlit.io/.

[41] «Panel - Documentation,» [Ηλεκτρονικό]. Available: https://panel.holoviz.org/.

[42] «Gradio - Documentation,» [Ηλεκτρονικό]. Available: https://gradio.app/docs/.

[43] «PySimpleGUI - Documentation,» [Ηλεκτρονικό]. Available: https://pysimplegui.readthedocs.io/en/latest/.

[44] «Tkinter - Documentation,» [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/tkinter.html.

[45] Λ. Φραγκουλόπουλος, «Διαδραστικές ασκήσεις Ψηφιακών Επικοινωνιών στη γλώσσα Python», προς υποβολή.

[46] «Custom Javascript - Documentation,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/advanced/html.html.

[47] «Markdown - Documentation,» [Ηλεκτρονικό]. Available: https://www.markdownguide.org/getting-started/.

[48] «Myst Markdown - Documentation,» [Ηλεκτρονικό]. Available: https://myst-parser.readthedocs.io/en/latest/syntax/typography.html.

[49] «Jupyter Notebooks Files - Documentation,» [Ηλεκτρονικό]. Available: https://docs.jupyter.org/en/latest/.

[50] «Jupyter Book - Configuration File,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/customize/config.html.

[51] «Jupyter Book - Table of Contents,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/structure/configure.html.

[52] C. J. -. Documentation, «Custom Javascript - Documentation,» [Ηλεκτρονικό]. Available: https://jupyterbook.org/en/stable/advanced/html.html.

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, περιγράφονται:

- η διαδικασία εγκατάστασης της Python,
- η εγκατάσταση των απαραίτητων βιβλιοθηκών Python με Conda,
- η εγκατάσταση της βιβλιοθήκης Jupyter Book,
- η δημιουργία ενός Jupyter Book,
- η δημοσίευσή του μέσω Gitub.

Εγκατάσταση της Python στον Υπολογιστή

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

- Κατεβάστε το εκτελέσιμο αρχείο της Python από την επίσημη ιστοσελίδα: https://www.python.org/downloads/
- Ακολουθήστε τον οδηγό εγκατάστασης και κατά την εγκατάσταση, βεβαιωθείτε ότι έχετε επιλέξει την επιλογή "Add Python to PAT".
- 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

2. Ακολουθήστε τον οδηγό εγκατάστασης.

3. Δημιουργία περιβάλλοντος Conda:

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

```
conda activate myenv
```

4. Εγκατάσταση του **Jupyter Notebook**:

```
conda install -c conda-forge notebook
```

5. Εγκατάσταση βιβλιοθηκών για επιστημονικούς υπολογισμούς:

```
conda install -c anaconda numpy
```

```
conda install -c anaconda scipy
```

```
conda install veeresht::scikit-commpy
```

6. Εγκατάσταση βιβλιοθηκών για γραφήματα:

```
conda install -c conda-forge matplotlib
```

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

```
conda install anaconda::ipython
```

```
conda install anaconda::ipywidgets
```

8. 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
   ```

2. Εντολές κατασκευής και προβολής του Jupyter Book:

   i. Εντολή κατασκευής του Jupyter Book:

   ```
   jupyter-book build mybook/
   ```

   ii. Για την προβολή του βιβλίου τοπικά, μεταβείτε στον φάκελο *_build/html* και ανοίξτε το αρχείο *index.html*.

   iii. 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 ανεβάσετε το βιβλίο στο Github και να το δημοσιεύσετε, ακολουθήστε τα παρακάτω βήματα:

1. Δημιουργία Github Repository:
    i. Συνδεθείτε στον λογαριασμό σας στο Github και δημιουργήστε ένα νέο repository.
    ii. Κατεβάστε το Git από την επίσημη ιστοσελίδα:

    https://git-scm.com/downloads

    iii. Ακολουθήστε τον οδηγό εγκατάστασης.
    iv. Ακολούθως, στο τερματικό του τοπικού σας συστήματος, από κάποιο φάκελο της επιλογής σας:

    ```
    git init
    ```

    ```
    git add
    ```

    ```
    git commit -m "Initial
    ```

    ```
    git remote add origin <URL_του_Repository>
    ```

    ```
    git push -u origin master
    ```

2. Ακολουθήστε τον επίσημο οδηγό δημιουργίας Github Pages:

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

3. 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.

4. Διανομή του Βιβλίου:
    i. Μετά τη δημοσίευση, το βιβλίο θα είναι διαθέσιμο στη διεύθυνση https://<όνομα_χρήστη>.github.io/<repository_name>.

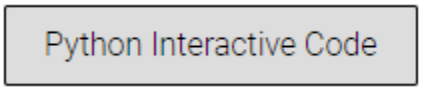
Παράρτημα ΙΙ

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.

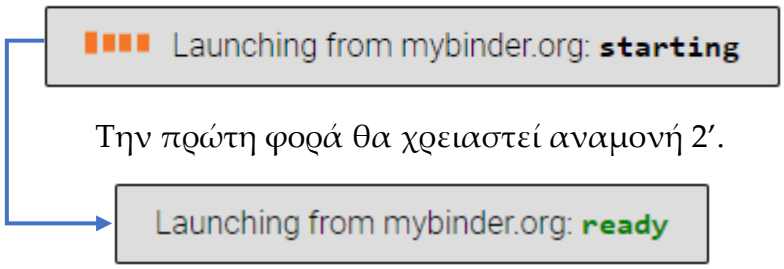
Εκκίνηση Διαδραστικού Κώδικα

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



Σχήμα 29: Κουμπί Interactive code

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

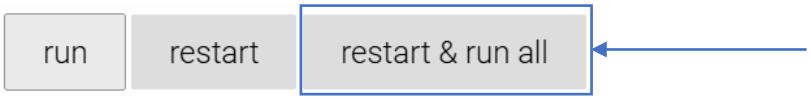


Σχήμα 30: Αναμονή για έτοιμο kernel

- Αν δείτε ότι η διαδικασία καθυστερεί για περισσότερο 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 την ανάλυση των εργαστηριακών ασκήσεων.

Documentation: Basic Elements of a Jupyter Book

Δομή του Jupyter Book

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

Βασικά Στοιχεία:

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

Αρχεία Περιεχομένου

- **Markdown** (.md) [47]:

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

- **MyST Markdown** (.md) [48]:

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

- **Jupyter Notebooks** (.ipynb) [49]:

  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 [50]:

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

```
#######################################################################################
# A default configuration that will be loaded for all jupyter books
# Users are expected to override these values in their own `_config.yml` file.
# This is also the "master list" of all allowed keys and values.
#######################################################################################

# Book settings

# The title of the book. Will be placed in the left navbar.
title: My Jupyter Book

# The author of the book
author: The Jupyter Book community

# Copyright year to be placed in the footer
copyright: "2023"

# A path to the book logo
logo: ""

# Patterns to skip when building the book. Can be glob-style (e.g. "*skip.ipynb")
exclude_patterns: [_build, Thumbs.db, .DS_Store, "**.ipynb_checkpoints"]

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

#######################################################################################
# Execution settings

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

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

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

```yaml
# The maximum time (in seconds) each notebook cell is allowed to run
  timeout: 30

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

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

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

#################################################################################
# Parse and render settings

# MyST extensions to enable (list of extensions)
parse:
  myst_enable_extensions:
    # Enables colon fences for better block delimitation
    - colon_fence
    # Enables the use of LaTeX-style math expressions
    - dollarmath
    # Converts bare links into proper hyperlinks
    - linkify
    # Allows using text substitution in markdown
    - substitution
    # Adds checklists to the Markdown files
    - tasklist

# URI schemes that will be recognized as external URLs in Markdown links
  myst_url_schemes: [mailto, http, https]

# Allows display math ($$) within inline contexts
  myst_dmath_double_inline: true

#################################################################################
# HTML-specific settings

# Path to a favicon image for the website
html:
  favicon: ""

# Whether to add an "edit this page" button to pages
  use_edit_page_button: false
```

```yaml
# Whether to add a repository link button
  use_repository_button: false

# Whether to add an "open an issue" button
  use_issues_button: false

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

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

# Whether to include the home page in the left Navigation Bar
  home_page_in_navbar: true

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

# Analytics settings for the website (Google Analytics, Plausible Analytics, etc.)
  analytics:
    plausible_analytics_domain: ""
    plausible_analytics_url: "https://plausible.io/js/script.js"
    google_analytics_id: ""

# Enable comments on the site (via Hypothesis or Utterances)
  comments:
    hypothesis: false
    utterances: false

# A banner announcement at the top of the site
  announcement: ""

#######################################################################################
# LaTeX-specific settings

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

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

#######################################################################################
# Launch button settings

# The interface for interactive links ("classic", "jupyterlab")
launch_buttons:
  notebook_interface: classic
```

```yaml
# The URL of the BinderHub (for executing code in the cloud)
  binderhub_url: ""

# The URL of the JupyterHub (for interactive computing)
  jupyterhub_url: ""

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

# The URL of Google Colab for cloud execution
  colab_url: ""

# The URL of Deepnote for interactive notebooks
  deepnote_url: ""

#######################################################################################
# Repository settings

# The URL to your book's repository
repository:
  url: https://github.com/executablebooks/jupyter-book

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

# The branch of the repository to use when creating links
  branch: master

#######################################################################################
# Advanced and power-user settings

# Additional Sphinx extensions to load (in addition to defaults)
sphinx:
  extra_extensions: []

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

# Whether to overwrite or recursively update the Sphinx configuration
  recursive_update: false

# Key-value pairs to directly override Sphinx configuration options
  config: {}
```

_toc.yml [51]:

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

```yaml
# Table of contents
# Learn more at https://jupyterbook.org/customize/toc.html

format: jb-book
root: intro
parts:
- caption: About
  chapters:
  - file: digital_communications_intro
- caption: Lab Exercises
  chapters:
  - file: content/Digcom_Lab1
  - file: content/Digcom_Lab2
  - file: content/Digcom_Lab3
  - file: content/Digcom_Lab4
  - file: content/Digcom_Lab5
  - file: content/Digcom_Lab6
- caption: Bit Error Rate Tool
  chapters:
  - file: content/Digcom_Lab_BerTool

```

Σχήμα 31: _toc.yml

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

Διαδραστικότητα με το Thebe

Μία 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.

- Binder _config.yml

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

```
launch_buttons:
        thebe: true
```

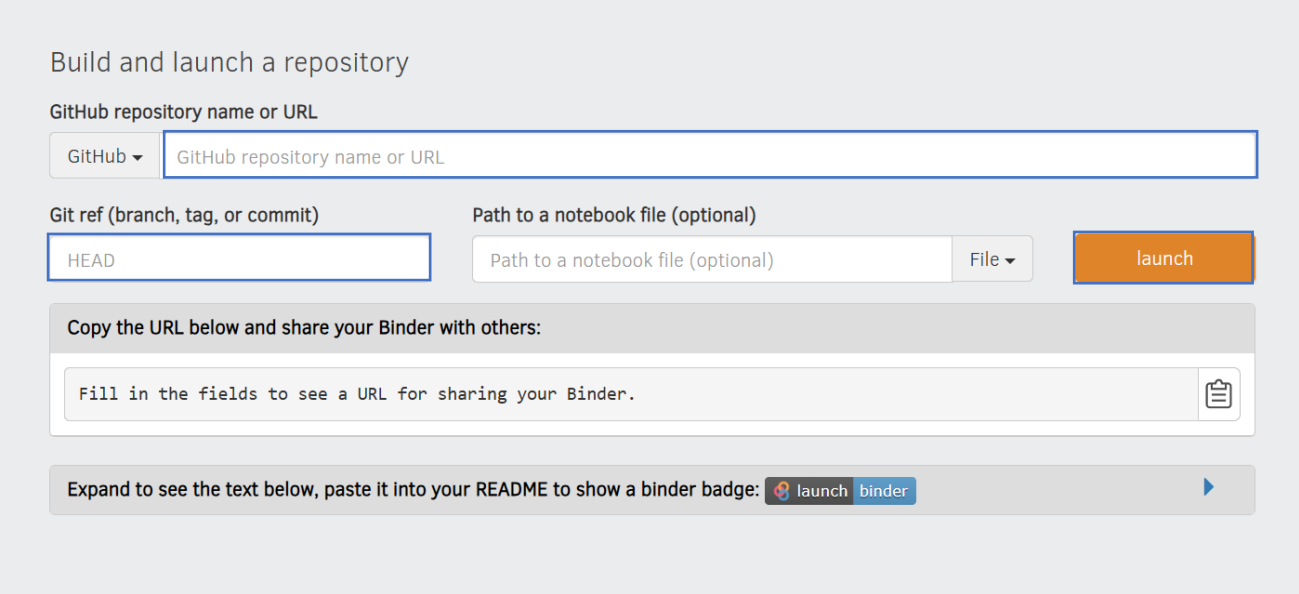
- requirements.txt

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

```
numpy==1.24.3
scipy==1.10.1
matplotlib
ipython
ipywidgets==7.7.2
requests
scikit-commpy
```

- Binder

Αφού συμπληρωθεί το αρχείο "requirements.txt", μεταβαίνουμε στην ιστοσελίδα https://mybinder.org/ . Εκεί, συμπληρώνουμε τα απαιτούμενα πεδία για το Github repository και πατάμε το κουμπί "launch".:



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

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.

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()
```

Επιστροφή στην υλοποίηση

Προσθήκη θορύβου:

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

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()
```

Επιστροφή στην υλοποίηση

Διαδραστική επεξεργασία μεταβλητών σήματος:

Με τη χρήση διαδραστικών εργαλείων, 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 απόκρισής τους με μορφή stem plot.

Κώδικας – Chapter 1 - Filter Frequency Response

```
# 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'Spectral density 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 απόκρισής τους τόσο στη συχνότητα όσο και στη 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')

# ---- 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()

# Example signal & sampling frequency
s  = np.random.randn(5000)
Fs = 8192                       # [Hz]

# ---- Update plots whenever f1 / f2 change -------------------------------
def update_plots(change):
    f1, f2 = f1_input.value, f2_input.value

    # Validation
    if not (0 <= f1 < f2 <= 0.48*Fs):
        with output_area:
            output_area.clear_output(wait=True)
            display(widgets.HTML(
                "<div style='color:black; background:#ffffcc; padding:10px; "
                "border-radius:5px; font-size:16px; text-align:center;'>"
                "<b>⚠ Warning:</b> 0 &lt; f1 &lt; f2 &lt; 0.48·Fs  (≈ 3900
Hz)</div>"))
        return

    # ---- Parks–McClellan (Equiripple) design --------------------------------
    numtaps = 129
    bands   = [0, f1*0.95,
           f1*1.05, f2*0.95,
           f2*1.05, Fs/2]
    desired = [0, 1, 0]          # stop, pass, stop
    weights = [1, 1, 1]
```

```
    hbp_pm = remez(numtaps, bands, desired, weight=weights, fs=Fs)

    # ---- Plots ------------------------------------------------------------
    with output_area:
        output_area.clear_output(wait=True)

        w, h = freqz(hbp_pm, worN=4096, fs=Fs)
        fig, axs = plt.subplots(1, 3, figsize=(14, 4))

        # (1) Impulse response
        axs[0].plot(hbp_pm, label='Equiripple BP')
        axs[0].set_title('Impulse Response')
        axs[0].set_xlabel('Sample')
        axs[0].set_ylabel('Amplitude')
        axs[0].legend()
        axs[0].grid(True)

        # (2) Magnitude response (dB) – ζουμ στη ζώνη αποκοπής
        axs[1].plot(w, 20*np.log10(np.abs(h)))
        axs[1].set_title('Frequency Response (Equiripple)')
        axs[1].set_xlabel('Frequency [Hz]')
        axs[1].set_ylabel('Magnitude [dB]')
        axs[1].set_ylim([-120, 5])     # δείξε τα ripples
        axs[1].grid(True)

        # (3) Power Spectrum of filtered signal
        s_pm = np.convolve(s, hbp_pm, 'same')
        f, Pxx = welch(s_pm, Fs, 'flattop', 1024, scaling='spectrum')
        axs[2].semilogy(f, np.sqrt(Pxx))
        axs[2].set_title('Power Spectrum (Welch)')
        axs[2].set_xlabel('Frequency [Hz]')
        axs[2].set_ylabel('Linear spectrum [V RMS]')
        axs[2].grid(True)

        plt.tight_layout()
        plt.show()

# ---- Connect callbacks & display UI ----------------------------------------
f1_input.observe(update_plots, names='value')
f2_input.observe(update_plots, names='value')

ui = widgets.HBox([widgets.VBox([f1_input, f2_input])],
                  layout=Layout(align_items='center'))
display(widgets.VBox([ui, output_area]))

# Initial render
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 ιστογράμματος 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 M-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):
    # 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]
    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:',style={'description_width': 'initial'})
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
        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) / (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')

# Set up the UI layout
inputs1 = widgets.HBox([nsamp_dropdown])
inputs12 = widgets.HBox([inputs1], 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='Bitstream 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 Dropdown, interact

# ——————————————————————————————————————————————————————————————
# Gray-code helper
# ——————————————————————————————————————————————————————————————
def gray_code(n: int) -> int:
    """Return the n-th Gray code word (integer)."""
    return n ^ (n >> 1)

# ——————————————————————————————————————————————————————————————
# Constellation generation (με Gray mapping)
# ——————————————————————————————————————————————————————————————
def generate_qam_constellation(L: int):
    """
    Δημιουργεί τα σημεία του L²-QAM και τα αντίστοιχα Gray labels.
    Επιστρέφει:
      mapping : 1-D array με συντεταγμένες (complex)
      labels  : λίστα δυαδικών strings μήκους 2·log2(L)
    """
    l = int(np.log2(L))                     # bits ανά διάσταση
    # Πίνακες Gray κωδίκων για I-άξονα (g) και Q-άξονα (g_rev)
    g      = [gray_code(i) for i in range(L)]
    g_rev  = g[::-1]

    # Πλέγμα συντεταγμένων
    x = np.arange(-(L - 1), L, 2)
    y = np.arange(-(L - 1), L, 2)
    xv, yv = np.meshgrid(x, y)
    mapping = (xv + 1j * yv).flatten()

    # Ετικέτες Gray: [g_Q][g_I]
    fmt = f'0{l}b'                          # zero-pad σε l bits
    labels = [
        format(g_rev[row], fmt) + format(g[col], fmt)
        for row in range(L) for col in range(L)
    ]
```

```
    return mapping, labels

# ──────────────────────────────────────────────────────────────
# Plot
# ──────────────────────────────────────────────────────────────
def plot_qam_constellation(L: int):
    mapping, labels = generate_qam_constellation(L)
    l = int(np.log2(L))

    plt.figure(figsize=(10, 7))
    plt.scatter(mapping.real, mapping.imag)

    # Προβολή ετικετών (δείχνουμε μέχρι 16×16 για ευκρίνεια)
    if L <= 16:
        dx, dy = -0.5, 0.3
        for k, lbl in enumerate(labels):
            plt.text(mapping[k].real + dx, mapping[k].imag + dy,
                     lbl, bbox=dict(facecolor='red', alpha=0.5))

    plt.grid(True)
    plt.xlim(-L, L)
    plt.ylim(-L, L)
    plt.title(f"L²-QAM Constellation (L = {L} → {L*L}-QAM)")
    plt.xlabel("In-phase")
    plt.ylabel("Quadrature")
    plt.show()

# ──────────────────────────────────────────────────────────────
# Interactive widget
# ──────────────────────────────────────────────────────────────
L_dropdown = Dropdown(options=[2, 4, 8, 16, 32, 64],
                      value=8, description='L^2-QAM:',
                      continuous_update=False)

interactive_plot = interact(plot_qam_constellation, L=L_dropdown)
```

Επιστροφή στην υλοποίηση

Κώδικας – Chapter 5 - PSK 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 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 τη 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
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.ylim(1e-8, 1e-1)
    plt.xlim(0.5, 10)
    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 είναι 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 τη 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 τον αριθμό 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='M-FSK',
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