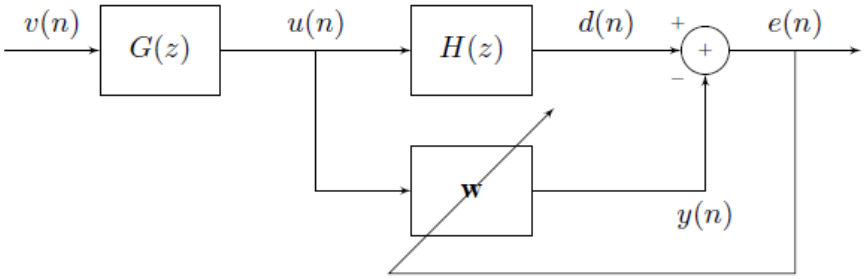
ΨΗΦΙΑΚΑ ΦΙΛΤΡΑ

Εργασία 2

Προσαρμογή στο Πεδίο της Συχνότητας

Μάστορας Ραφαήλ-Ευάγγελος
ΑΕΜ: 7918
25/4/2017



$$u(n) = -0.34\,u(n-1) + v(n)$$

Ερώτημα α)

Στο πρώτο ερώτημα μας ζητήθηκε να επιβεβαιώσουμε τους υπολογισμούς του ταχέως μετασχηματισμού Fourier (FFT). Για να το πετύχω αυτό χρησιμοποίησα την απόδειξη :

$$\hat{x}_k = \sum_{j=0}^{2^q-1} x_j \overbrace{e^{-2\pi i \frac{jk}{2^q}}}^{\omega_n^{jk}} = \sum_{j=0}^{2^{q-1}-1} x_{(2j)}\, e^{-2\pi i \frac{2jk}{2^q}} + \sum_{j=0}^{2^{q-1}-1} x_{(2j+1)}\, e^{-2\pi i \frac{(2j+1)k}{2^q}}$$

$$= \underbrace{\sum_{j=0}^{2^{q-1}-1} x_{(2j)} \overbrace{e^{-2\pi i \frac{jk}{2^{q-1}}}}^{\omega_{\frac{n}{2}}^{jk}}}_{\hat{x}_k^{(1)}} + \overbrace{e^{-2\pi i \frac{k}{2^q}}}^{\omega_n^k} \underbrace{\sum_{j=0}^{2^{q-1}-1} x_{(2j+1)} \overbrace{e^{-2\pi i \frac{jk}{2^{q-1}}}}^{\omega_{\frac{n}{2}}^{jk}}}_{\hat{x}_k^{(2)}}, \; k = 0, 1, \ldots, \frac{n}{2} - 1$$

Αφού βεβαίως την μετέτρεψα σε κώδικα matlab. Η υλοποίηση αυτή βρίσκεται στο αρχείο fftproof.m

Ερώτημα β)

Το ζητούμενο του ερωτήματος αυτού ήταν να δημιουργήσουμε αναδρομική συνάρτηση σε matlab για τον υπολογισμό του FFT για οποιαδήποτε ακολουθία της οποίας το μήκος είναι δύναμη του δύο. Επιπρόσθετα ζητήθηκε να υπολογίσουμε το υπολογιστικό κόστος του FFT καθώς και της αναδρομικής μας σχέσης, αλλά και να επιβεβαιώσουμε τα αποτελέσματα της αναδρομικής σχέσης που δημιουργήσαμε.

Ο κώδικας της αναδρομικής συνάρτησης βρίσκεται στο αρχείο fftrecursive.m . Η συνάρτηση αυτή δέχεται ως όρισμα μια οποιαδήποτε ακολουθεία μήκους 2^q και επιστρέφει τον μετασχηματισμό Furier της, ενώ παράλληλα δύο counter (countMuls και countSums) κρατάνε τον αριθμό των πολλαπλασιασμό και των προσθέσεων σύμφωνα με τους τύπους :
countMuls=countMuls+n/2;
countSums=countSums+2*length(Yo);
Αυτό γιατί σε κάθε αναδρομική κλήση της συνάρτησης εκτελούνται n/2 πολλαπλασιασμοί κατά την εύρεση της Yo αλλά και 2*length(Yo) προσθέσεις, κατά την εύρεση του τελικού Y.
Επίσης για την περίπτωση ακολουθιών μήκους 2, καθώς T(2)=4 και τα flops της μιγαδικής πρόσθεσης ισοδυναμούν με 2, χρησιμοποιήθηκε το countSums= countSums+2.

Έπειτα για τον υπολογισμό του υπολογιστικού κόστους δημιουργήθηκε η αναδρομική συνάρτηση T(n) που υπολογίζει το κόστος του FFT σύμφωνα με τον παρακάτω τύπο, ο οποίος μετά απο πράξεις μετατράπηκε σε T(n)=5*n+2*T(n/2), καθώς T(2)=4.

$$T(n) = \frac{n}{2}T(2) + 6\frac{n}{2} + 2T\left(\frac{n}{2}\right)$$

Τέλος δημιούργησα ενα αρχείο complexityCalculation.m το οποίο καλεί την αναδρομική συνάρτηση fftrecursive.m , βρίσκει τον FFT με βάση την built in function του matlab fft() και συγκρίνει τα αποτελέσματα καθώς και το υπολογιστικό κόστος. Το υπολογιστικό κόστος και στις δύο περιπτώσεις υπολογίστηκε ίσο για διαφόρων μήκων ακολουθίες. Συγκεκριμένα για μήκος ακολουθίας 2^3 υπολογίστηκε 96 flops και το error μεταξύ της συνάρτησης fft() και της αναδρομικής που δημιούργησα ίσο με 8.881784e-16.

Ερώτημα γ)

Για το ερώτημα αυτό χρειάστηκε να δημιουργήσω το αρχείο convolutionTheorem.m μέσω του οποίου υπολογίζεται η συνέλιξη τυχαίων σημάτων x,y με τις τέσσερεις μεθόδους που ζητήθηκαν.

i)Για τη πρώτη μέθοδο το μόνο που χρειάστηκε ήταν η built in συνάρτηση conv(x,y) και αποτελεί τη βάση αναφοράς για τα αποτελέσματα.

ii)Για την δεύτερη μέθοδο χρησιμοποιήθηκε η συνάρτηση
`toeplitz([y;zeros(length(x)-1,1)],[y(1);zeros(length(x)-1,1)])`
προκειμένου να βρεθεί ο πίνακας Toeplitz πίνακα **Y** του σήματος y. Έπειτα ο πίνακας αυτός πολλαπλασιαζόμενος με το σήμα x μας δίνει την συνέλιξη των x και y.

iii)Στην τρίτη μέθοδο υπολογίστηκε ο τετραγωνικός Circulan πίνακας **C** του πίνακα **Y**. Για να το πετύχω αυτό έκανα διαδοχικές κατάλληλες μεταθέσεις των στοιχείων της πρώτης στήλης του πίνακα **Y**. Προκειμένου να πάρουμε την συνέλιξη των σημάτων με την μέθοδο αυτή πολλαπλασιάζουμε τον πίνακα **C** με έναν κατάλληλα padded πίνακα **x**.

iv)Τέλος η τέταρτη μέθοδος υλοποιήθηκε με την τις συναρτήσεις fft() και ifft() με την εντολή
`ifft(fft(x, length(x)+length(y)-1).*(fft(y, length(x)+length(y)-1)))`
Που στην ουσία υπολογίζει τον πολλαπλασιασμό των σημάτων x και y στο πεδίο της συχότητας και έπειτα επιστρέφει το αποτέλεσμα στο πεδίο του χρόνου σύμφωνα με την σχέση

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

Τα αποτελέσματα της σύγκρισης για σήματα μήκους 2^10 :

Error between Toeplitz method and conv(): 2.779724e-12
Error between Circulant method and conv(): 2.779724e-12
Error between FFT method and conv(): 2.962169e-12

Ενώ το σφάλμα μειώνεται ακόμα περισσότερο για μικρότερου μήκους σήματα.

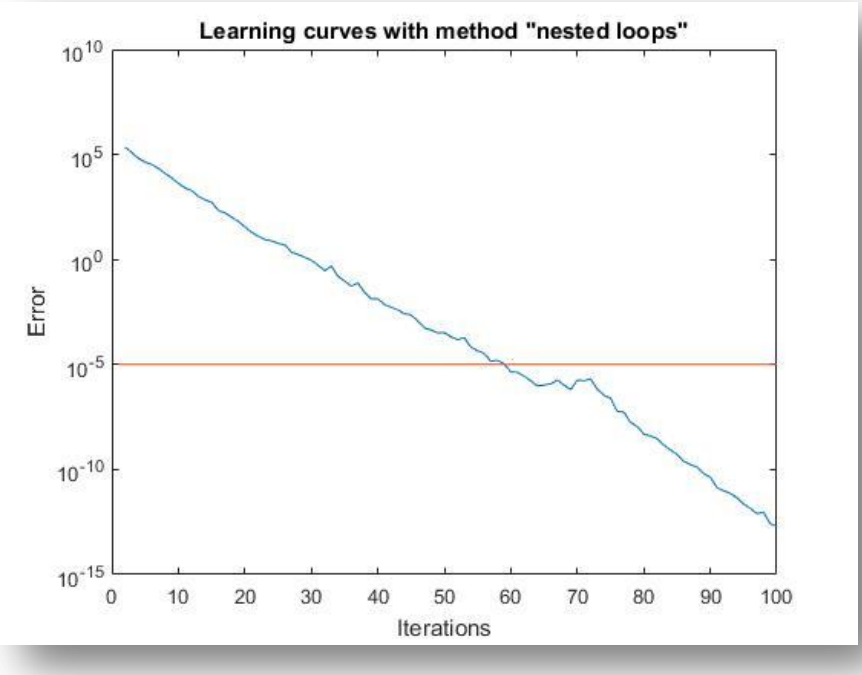
Ερώτημα δ)

Για το ερώτημα αυτό υλοποιήθηκαν τέσσερεις παραλαγές του αλγορίθμου Block LMS η υλοποίηση των οποίων βρίσκεται στο αρχείο BlockLMSmethods.m . Τα παρακάτω αποτελέσματα είναι για 2^{10} συντελεστές και μήκος σήματος 102400 δείγματα. Επίσης χρησιμοποιήθηκε mu=6.0698e-04 για τις τρείς πρώτες μεθόδους και mu=3.7e-04 για την Uncostrained μέθοδο, προκειμένου όλες οι μέθοδοι να χρειάζονται λιγότερες επαναλήψεις μέχρι να φτάσουν σε σφάλμα μικρότερο του 10^{-5}. Το mu βρέθηκε για την κάθε μέθοδο μετά απο δοκιμές.

i. Με δύο εμφωλευμένους βρόχους (nested loops)

Η υλοποίηση αυτή είναι η πιο απλουστευμένη απο τις υπόλοιπες τέσσερεις και χρησιμοποιεί 2 εμφωλευμένους βρόγχους για την μοντελοποίηση του αγνώστου συστήματος. Η υλοποίηση μου ακολουθεί τον αλγόριθμο που φαίνεται παρακάτω.

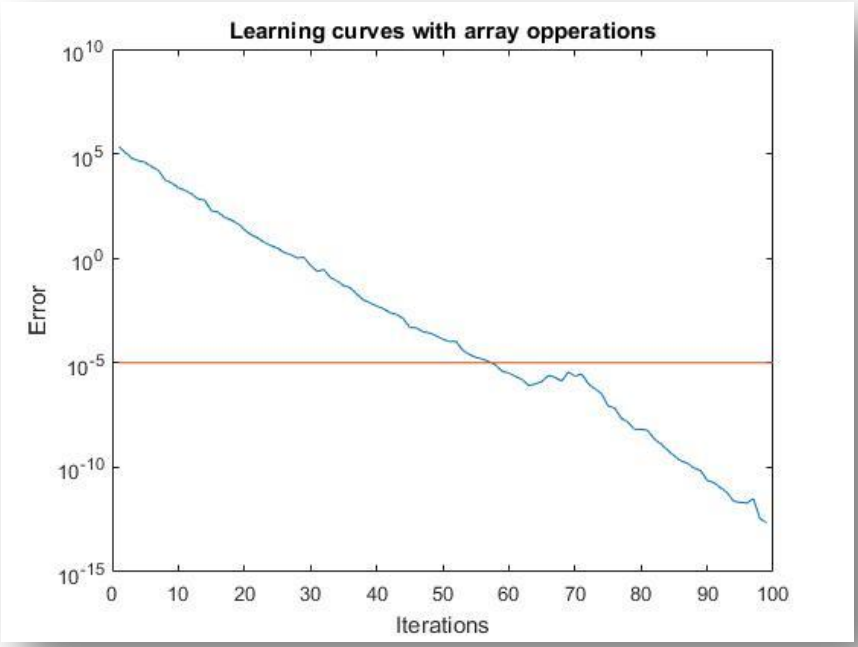
1 Initialize the algorithm with an arbitrary parameter vector $\underline{w}(0)$, for example $\underline{w}(0) = 0$.
2 Iterate for $k = 0, 1, 2, 3, \ldots, k_{max}$ (k is the block index)
2.0 Initialize $\underline{\phi} = 0$
2.1 Iterate for $i = 0, 1, 2, 3, \ldots, (L-1)$
2.1.0 Read /generate a new data pair, $(\underline{u}(kL+i), d(kL+i))$
2.1.1 (Filter output) $y(kL+i) = \underline{w}(k)^T \underline{u}(kL+i) = \sum_{j=0}^{M-1} w_j(k) u(kL+i-j)$
2.1.2 (Output error) $e(kL+i) = d(kL+i) - y(kL+i)$
2.1.3 (Accumulate) $\underline{\phi} \leftarrow \underline{\phi} + \mu e(kL+i)\underline{u}(kL+i)$
2.2 (Parameter adaptation) $\underline{w}(k+1) = \underline{w}(k) + \underline{\phi}$
□
Complexity of the algorithm: $2M+1$ multiplications and $2M + \frac{M}{L}$ additions per iteration



Με την μέθοδο αυτή φαίνεται πως το τελικό σφάλμα είναι εξαιρετικά μικρό της τάξης $10^{-13.}$ Ένα μέσο επιθυμητό σφάλμα της τάξης 10^{-5} μπορεί να επιτευχθεί μετά απο 59 επαναλήψεις, ενώ ο χρόνος εκτέλεσης του αλγορίθμου αυτού ήταν 3.0051 δευτερόλεπτα.

ii. Με ένα βρόχο και πράξεις πινάκων

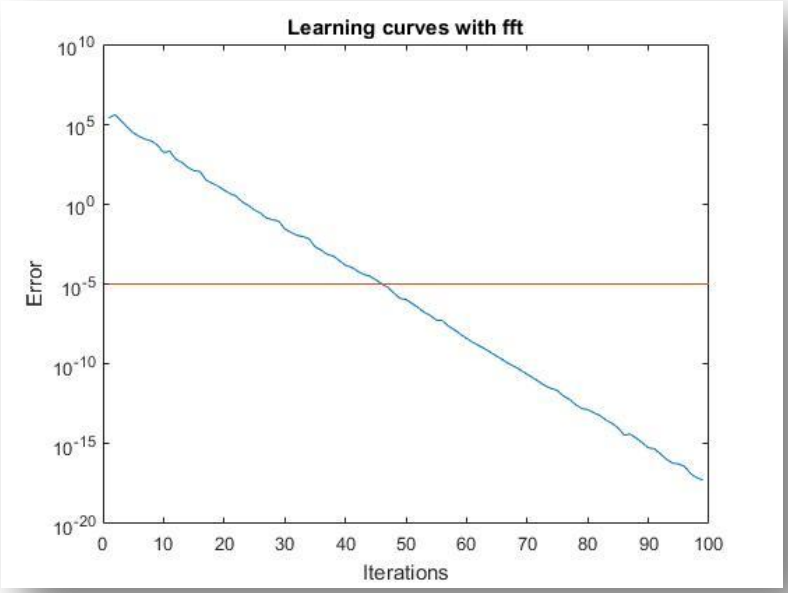
Η υλοποίηση αυτή αποτελεί μια βελτίστοποίηση της προηγούμενης μεθόδο με την χρήση πράξεων πινάκων αντί των εμφολευμένων βρόγχων επανάληψης.



Και η μέθοδος αυτή έχει εξαιρετικά μικρό σφάλμα της τάξης 10^{-13}. Θεωρώντας πάλι ανεκτό σφάλμα της τάξης 10^{-5} , αυτό μπορεί μπορεί να επιτευχθεί μετά απο 58 επαναλήψεις, ενώ ο χρόνος εκτέλεσης του αλγορίθμου αυτού ήταν 1.5491 δευτερόλεπτα, δηλαδή μισός από την μέθοδο με τις εμφωλευμένους βρόγχους. Αυτό ήταν αναμενόμενο καθώς οι εμφωλευμένοι βρόγχοι εισάγουν πολυπλοκότητα και επομένως καθυστέρηση στην εκτέλεση, ενώ από τη άλλη το Matlab είναι υλοποιημένο για να κάνει πολύ γρήγορα πράξεις πινάκων.

iii. Με προσαρμογή στο πεδίο της συχνότητας κάνοντας χρήση του FFT.

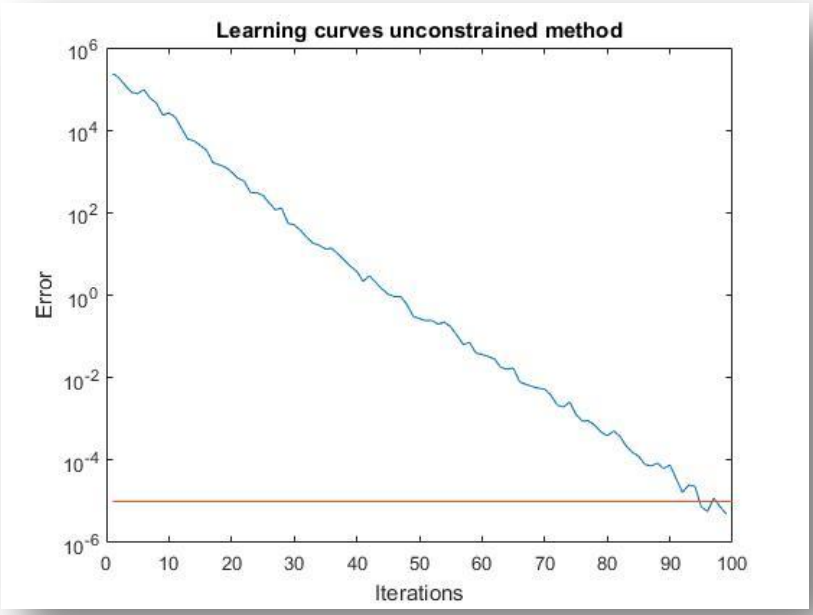
Η συγκεκριμένη υλοποίηση κάνει χρήση των γρήγορων μετασχηματισμών Fourier (FFT) προκειμένου να γίνει η μοντελοποίηση του αγνώστου συστήματος. Για την υλοποίηση αυτή συμβουλεύτηκα το σχήμα σελίδα 17 απο το BlockLMS.pdf, απο τις σημειώσεις του μαθήματος.



Η συγκεκριμένη μέθοδος έχει ακόμα μικρότερο σφάλμα, της τάξης 10^{-17}. Μετά απο 46 επαναλήψεις ο αλγόριθμος έχει μειώσει το σφάλμα σε 10^{-5}, ενώ ο χρόνος εκτέλεσης του αλγορίθμου αυτού ήταν 0.0340 δευτερόλεπτα, δηλαδή 45 φορές πιο γρήγορος απο την μέθοδο με πράξεις πινάκων και 88 φορές πιο γρήγορος απο την μέθοδο με δύο εμφωλευμένους βρόγχους. Αυτή η διαφορά στην ταχύτητα είναι αναμενόμενη καθώς ο Fast Block LMS έχει μειωμένη πολυπλοκότητα σε σχέση με τον Block LMS. Ασφαλώς η διαφορά ταχύτητας θα ήταν ακόμα μεγαλύτερη αυξανομένου του μήκους σήματος και του αριθμού συντελεστών.

iv. Με μη περιορισμένη (unconstrained) προσαρμογή στο πεδίο της συχνότητας.

Η υλοποίηση αυτή είναι παρόμοια με την προηγούμενη, χωρίς όμως να επιβάλλονται ορισμένοι περιορισμοί της προηγούμενης υλοποίησης. Συγκεκριμένα απο το σχήμα που αναφέρθηκε πριν, αφαιρείται το κομμάτι που βρίσκεται ανάμεσα στις διακεκομμένες γραμμές. Δηλαδή παραλείπεται η αφαίρεση των τελευταίων M στοιχείων απο τον πίνακα του αντίστροφου μετασχηματισμού Fourier και η αντικατάστασή τους απο μηδενικά. Η υλοποίηση αυτή αποτελεί μια άλλη έκδοση του Fast Block LMS, με ακόμα μικρότερη πολυπλοκότητα. Ο αλγόριθμος αυτός κάνει χρήση τριών μετασχηματισμών Fourier.



Παρατηρούμε πως η μέθοδος αυτή φτάνει ένα ελάχιστο σφάλμα της τάξης 10^{-5}, αρκετά μεγαλύτερο δηλαδή απο τα σφάλματα που επιτύγχαναν οι προηγούμενες υλοποιήσεις. Το σφάλμα αυτό επιτυγχάνεται μετά απο 95 επαναλήψεις, ωστόσο ο χρόνος εκτέλεσης είναι 0.0179 δευτερόλεπτα, μικρότερος απο όλες τις προηγούμενες υλοποιήσεις.

Παρά το γεγονός οτι η μέθοδος αυτή είναι η γρηγορότερη απο τις προηγούμενες με ακετά μεγάλη διαφορά εμφανίζει κάποια μειονεκτήματα. Όταν ο αριθμός των επεξεργασμένων block αυξάνεται, ο πίνακας βαρών δεν συγκλίνει πλέον στη λύση Wiener, ενώ επίσης το σφάλμα όπως είδαμε είναι αυξημένο σε σχέση με τους προηγούμενους αλγορίθμους.

Συνεπώς στην ερώτηση ποια μέθοδος είναι πιο συμφέρουσα υπολογιστηκά, θεωρώ πως η μέθοδος Fast Block LMS είναι η καλύτερη. Παρόλα αυτά αν το κύριο μέλημα μας είναι ο χρόνος εκτέλεσης, ενώ παράλληλα ο αριθμός των συντελεστών είναι μικρός, το μήκος του σήματος είναι μεγάλο και εφόσον δεν υπάρχει απαίτηση το σφάλμα να είναι το μικρότερο δυνατό, μπορούμε να χρησιμοποιήσουμε την unconstrained μέθοδο.