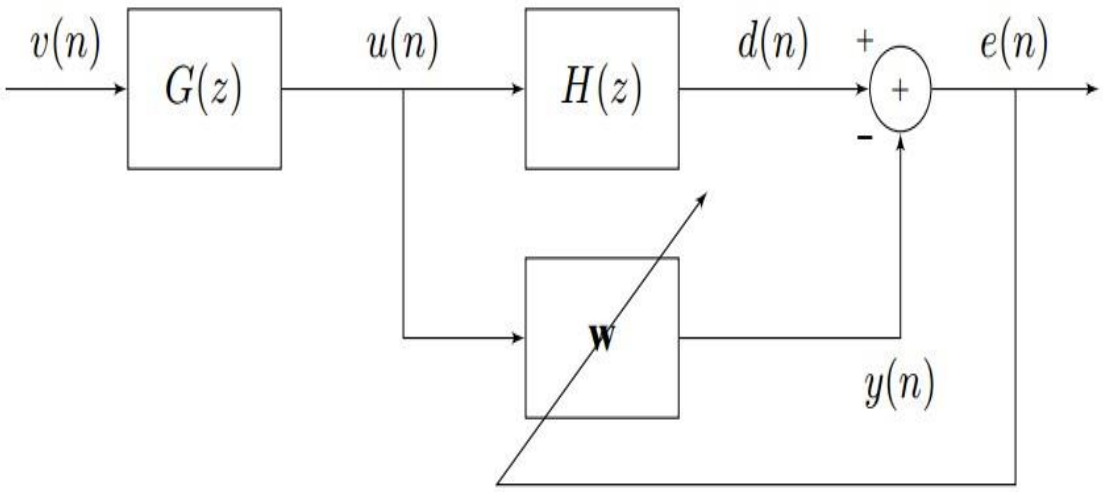
ΨΗΦΙΑΚΑ ΦΙΛΤΡΑ

ΕΡΓΑΣΙΑ 2

ΠΡΟΣΑΡΜΟΓΗ ΣΤΟ ΠΕΔΙΟ ΤΗΣ ΣΥΧΝΟΤΗΤΑΣ



ΘΕΟΧΑΡΗΣ ΤΡΙΑΝΤΑΦΥΛΛΙΔΗΣ

ΑΕΜ : 7995

α)

Αρχικά δόθηκε ο κώδικας fftproof ο οποίος συμπληρώθηκε έτσι ώστε να αποδειχτεί η ορθή χρήση του fft. Κάτι τέτοιο έγινε προσθέτοντας στην απόδειξη την παρακάτω γραμμή, μετατρέποντας την σε κώδικα matlab:

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

$\hat{x}_k^{(1)}$

που προέκυψε από

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

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

$\hat{x}_k^{(1)}$

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}^{(1)} + \Omega_n \hat{\mathbf{x}}^{(2)} \\ \hat{\mathbf{x}}^{(1)} - \Omega_n \hat{\mathbf{x}}^{(2)} \end{bmatrix}, \; \Omega_n = \begin{bmatrix} \omega_n^0 & 0 & \ldots & 0 \\ 0 & \omega_n^1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \omega_n^{k-1} \end{bmatrix}$$

Ο συμπληρωμένος κώδικας βρίσκεται στο επισυναπτόμενο αρχείο fftproof.m.

β)

- Αρχικά γράφτηκε αναδρομικός κώδικας ο οποίος υπολογίζει τον fast fourier transform μίας ακολουθίας μήκους δύο. Ο κώδικας αυτός δίνεται στο αρχείο fft_recursive. Δέχεται ως όρισμα μία ακολουθία x και επιστρέφει τον μετασχηματισμό fourier αυτής. Όσο να αφορά τον υπολογισμό των πολλαπλασιασμών και των προσθέσεων αυτού του αναδρομικού κώδικα, αυτό γίνεται με τη χρήση των γραμμών:

  ```
  muls=muls+n/2;
  sums=sums+2*length(Y2);
  ```

  καθώς σε κάθε αναδρομική κλήση της fft recursive εκτελούνται n/2 πολλαπλασιασμοί κατά την εύρεση της Y2 αλλά και 2*length(Y2) προσθέσεις, κατά την εύρεση του τελικού Y.

ΣΗΜΕΙΩΣΗ: *Κατά τον υπολογισμό των μιγαδικών πολλαπλασιασμών και προσθέσεων λαμβάνεται ειδική μεταχείριση των ακολουθιών μήκους 2, λόγω του γεγονότος ότι T(2)=4 και τα flops της μιγαδικής πρόσθεσης ισοδυναμούν με 2.*

- Στη συνέχεια συντάχθηκε η συνάρτηση T, η οποία χρησιμοποιεί την αναδρομική σχέση

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

έτσι ώστε να υπολογίσει το θεωρητικό υπολογιστικό κόστος του fft. Λαμβάνοντας υπόψην το γεγονός ότι T(2)=4, η παραπάνω σχέση μπορεί να γίνει $T(n)= \frac{n}{2} 4 + \frac{n}{2} 6 + 2T(\frac{n}{2}) = 5n + 2T(\frac{n}{2})$.

- Τελικά δημιουργήθηκε το script complexity_calc το οποίο θέτει ως είσοδο μία ακολουθία x (η ίδια που υπάρχει στο fftproof) και για αυτή καλείται η fft_recursive. Μέσα σε αυτό το script επιπλέον ελέγχεται εάν το θεωρητικό υπολογιστικό κόστος του fft μέσω την αναδρομικής σχέσης T, είναι ίδιο με το υπολογιστικό κόστος που υπολογίζεται από την fft_recursive. Ακόμη ελέγχεται το σφάλμα υπολογισμού του fft μέσω της αναδρομικής συνάρτησης, σε σχέση με τον fft του matlab. Πράγματι, τρέχοντας το script εκτυπώνεται το παρακάτω μήνυμα:

result confirmed with recursive fft algorithm error =: 7.691851e-16

γ)i)

x = [1 2 3];y = [-4 5 -6];

h= conv(x,y)

h= -4 -3 -8 3 -18

ii) Form `r` by padding `x` with zeros. The length of `r` is the convolution length `x + y - 1`.

```
r = [x zeros(1,length(y)-1)]
r =

  1     2     3     0     0
```

Form the column vector `c`. Set the first element to `x(1)` because the column determines the diagonal. Pad `c` because `length(c)` must equal `length(y)` for convolution.

```
c = [x(1) zeros(1,length(y)-1)]
c =

    1     0     0
```

Form the convolution matrix `xConv` using `toeplitz`. Then, find the convolution using `y*xConv`.

```
xConv = toeplitz(c,r)
xConv =

    1     2     3     0     0
    0     1     2     3     0
    0     0     1     2     3
 y*xConv

 ans = -4    -3    -8     3   -18
```

iii)

cconv(y,x)

ans =

-4.0000 -3.0000 -8.0000 3.0000 -18.0000

iv) x = [1 2 3];y = [-4 5 -6];

x1=[x 0 0]; y1=[y 0 0];ifft(fft(x1).*fft(y1))

-4.0000 -3.0000 -8.0000 3.0000 -18.0000

δ)

Τελικά μελετήθηκαν και υλοποιήθηκαν τέσσερις διαφορετικές εκδόσεις του Block LMS. Ο υλοποιήσεις αυτές βρίσκονται στα Block_LMSa, Block_LMSb, Block_LMSc και Block_LMSd και επεξηγούνται επαρκώς, όπου είναι απαραίτητο με σχόλια. Παρακάτω θα δοθούν για κάθε έναν αλγόριθμο κάποιες γενικές παρατηρήσεις ως προς την υλοποίηση, οι καμπύλες εκμάθησης καθώς και οι χρόνοι εκτέλεσης του κάθε ενός:

1) Υλοποίηση με δύο εμφωλευμένους βρόγχους:

ΠΑΡΑΤΗΡΗΣΕΙΣ:

Πρόκειται για την απλούστερη και μη βελτιστοποιημένη υλοποίηση του Block LMS. Για την δημιουργία αυτού του κώδικα χρησιμοποιήθηκε ο βασικός αλγόριθμος όπως δίνεται παρακάτω:

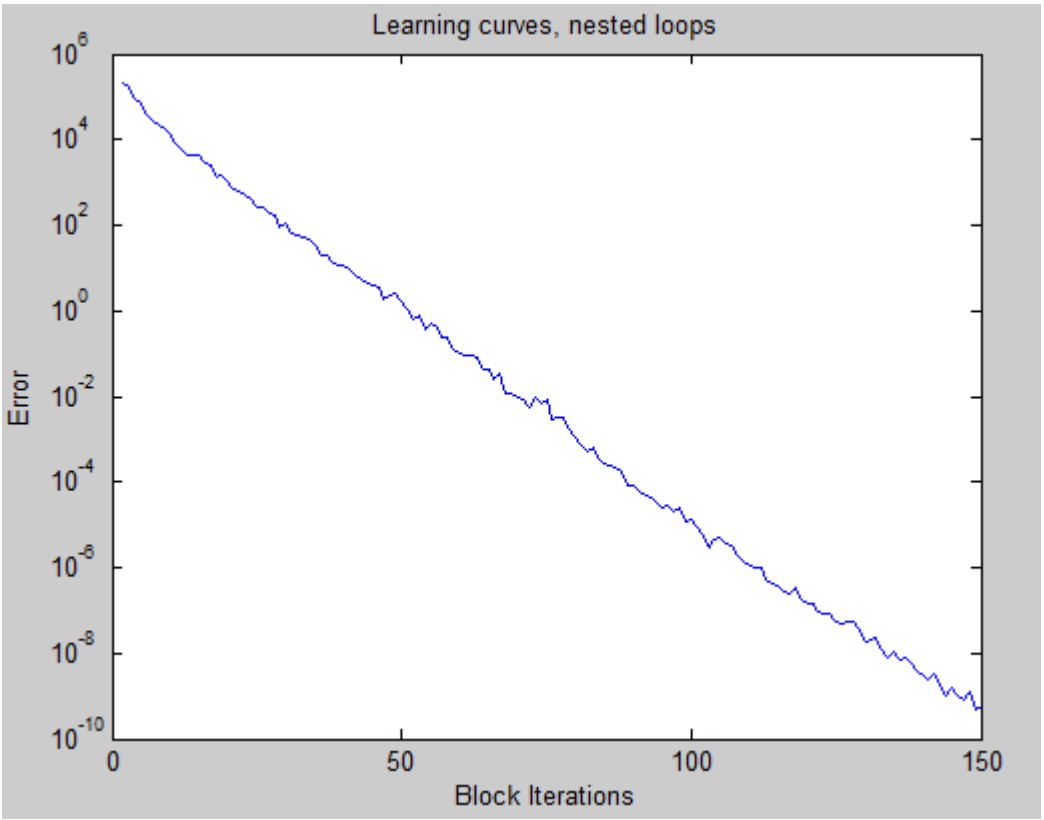
Given

- the (correlated) input signal samples $\{u(1), u(2), u(3), \ldots\}$, randomly generated;
- the desired signal samples $\{d(1), d(2), d(3), \ldots\}$ correlated with $\{u(1), u(2), u(3), \ldots\}$

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

Η μετατροπή της παραπάνω λεκτικής περιγραφής σε κώδικα matlab βρίσκεται στο αρχείο Block_LMSa.

ΚΑΜΠΥΛΕΣ ΕΚΜΑΘΗΣΗΣ:



Στο παραπάνω σχήμα φαίνεται ότι το τελικό σφάλμα κινείται σε μικρές τιμές της τάξης του 10^{-10}. Όσο να αφορά την ταχύτητα σύγκλισης, παρατηρείται ότι το σφάλμα τάξης 10^{-5} που μπορεί να θεωρηθεί ανεκτό, επιτυγχάνεται κοντά στην 90 επανάληψη.

ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ:

Εισάγοντας τα tic & toc που διαθέτει το matlab για τη μέτρηση χρόνου εκτέλεσης προγραμμάτων βρέθηκε ότι time= 12.2733s, για μήκος εισόδου u 153600, και μήκος φίλτρου 2^{10}συντελεστές. Ο μεγάλος αυτός χρόνος μπορεί να εξηγηθεί από την πολυπλοκότητα που εισάγει η διπλή for loop. ***ΣΗΜΕΙΩΣΗ:*** *Οι κώδικες που παραδόθηκαν διαθέτουν τρόπο υπολογισμού του συντελεστή μ, όμως για ταχύτητα υπολογισμών σε όλες τις συναρτήσεις έχει παρθεί το μ σταθερό και ίσο με 3.2e-04.*

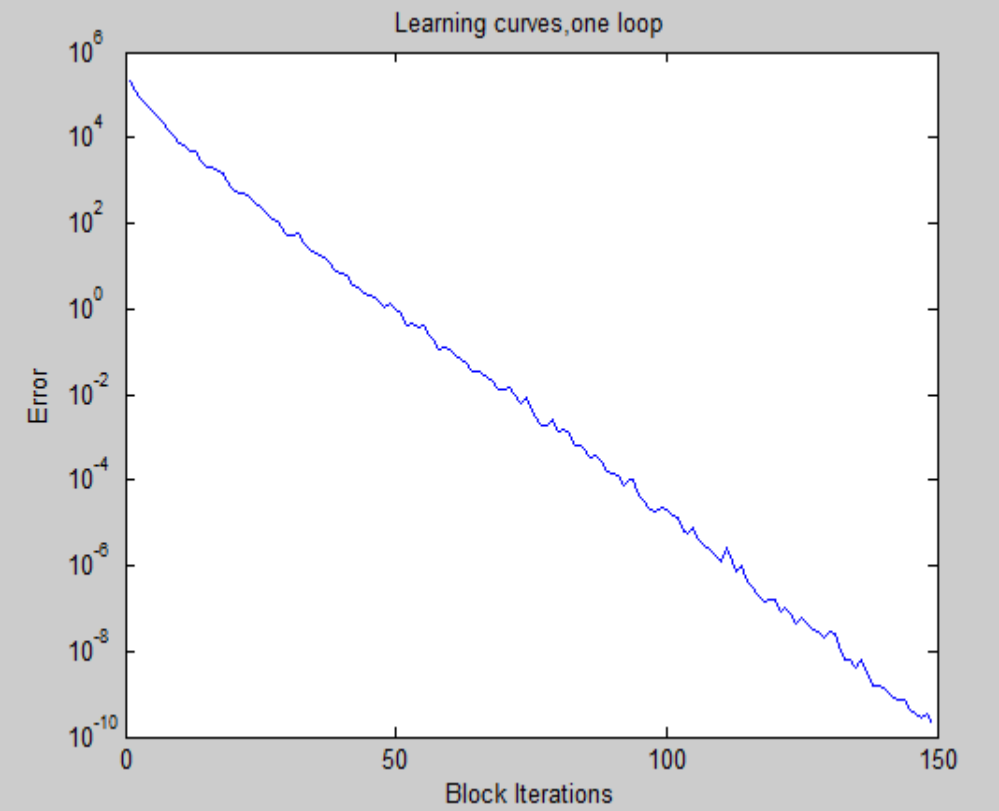
2) Υλοποίηση με έναν βρόγχο και πράξεις πινάκων:

ΠΑΡΑΤΗΡΗΣΕΙΣ:

Πρόκειται για μία βελτιστοποιημένη έκδοση της προηγούμενης υλοποίησης, προς αποφυγή της πολυπλοκότητας που εισάγει η χρήση της εμφωλευμένης for. Για την υλοποίηση αυτής της συνάρτησης χρησιμοποιήθηκε ο

αλγόριθμος block LMS όπως δόθηκε παραπάνω, με τη διαφορά ότι υπολογίστηκαν κάθε φορά είτε τα matrixes είτε τα vectros που ήταν απαραίτητα για την εύρεση της εξόδου του φίλτρου, του σφάλματος καθώς και του phi, χωρίς την χρήση της εσωτερικής for αλλά μόνο με πράξεις πινάκων.
Η μετατροπή της παραπάνω παραλλαγής σε κώδικα matlab βρίσκεται στο αρχείο Block_LMSb.

ΚΑΜΠΥΛΕΣ ΕΚΜΑΘΗΣΗΣ:



Στο παραπάνω σχήμα φαίνεται ότι το τελικό σφάλμα προσεγγίζει πολύ μικρές τιμές της τάξης του 10^{-10}. Θα έλεγε κανείς ότι δεν παρατηρείται μεγάλη διαφορά στην ταχύτητα σύγκλισης σε σχέση με το α ερώτημα. Δηλαδή και εδώ το σφάλμα τάξης 10^{-5} ,επιτυγχάνεται κοντά στην 90 επανάληψη.

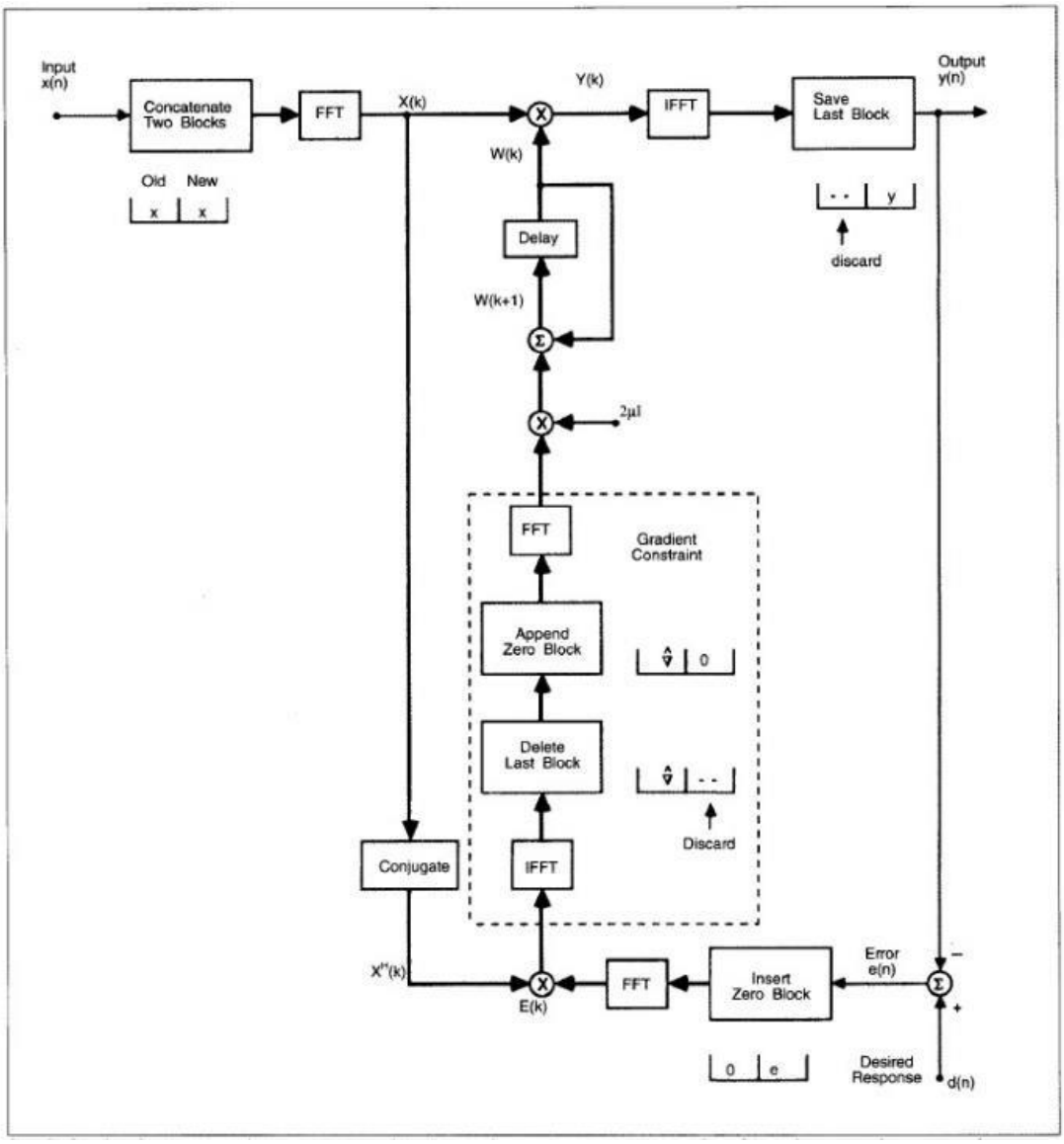
ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ:

Εισάγοντας τα tic & toc που διαθέτει το matlab για τη μέτρηση χρόνου εκτέλεσης προγραμμάτων βρέθηκε ότι time= 4.4061s, για μήκος εισόδου u 153600, και μήκος φίλτρου 1024 συντελεστές. Δηλαδή υπάρχει σημαντική βελτίωση της ταχύτητα σε σχέση με τον α τρόπο. Κάτι τέτοιο ήταν αναμενόμενο καθώς πλέον έχει αποφυγή της N^2 που εισάγει η εμφωλευμένη for loop.

3) Υλοποίηση με προσαρμογή στο πεδίο της συχνότητας:

ΠΑΡΑΤΗΡΗΣΕΙΣ:

Πρόκειται για μία υλοποίηση που κάνει εκτενή χρήση των μετασχηματισμών fft. Η υλοποίηση αυτή έγινε με βάση το παρακάτω σχήμα:

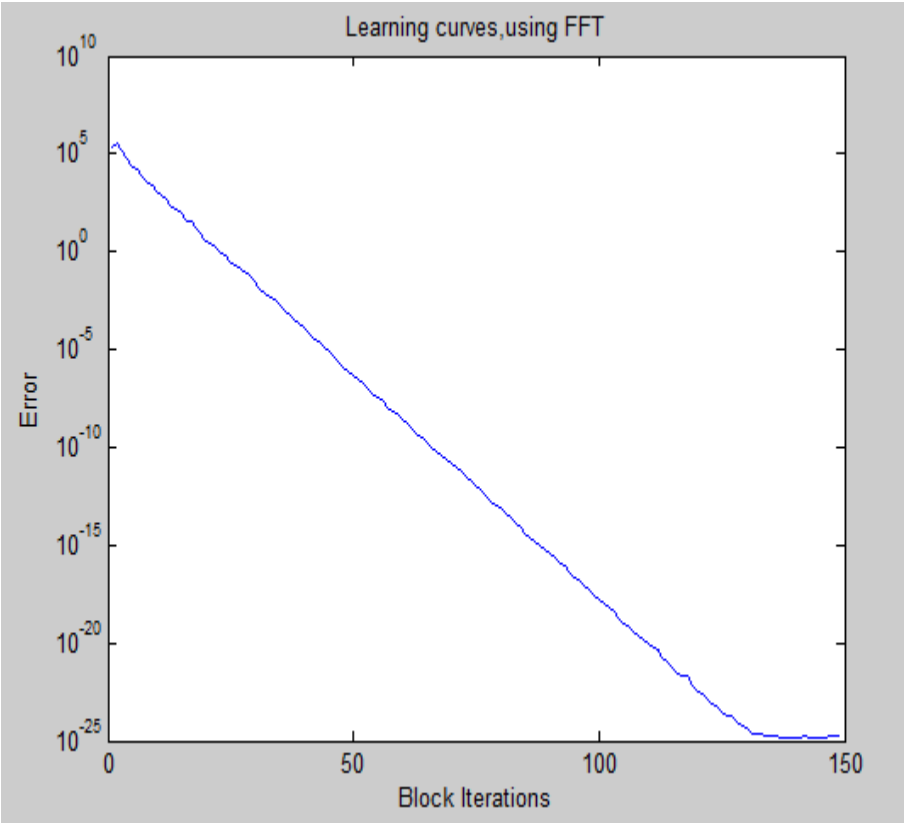


Η μετατροπή του παραπάνω σχήματος σε κώδικα βρίσκεται στο αρχείο Block_LMSc.

ΣΗΜΕΙΩΣΗ: *Για τον υπολογισμό του phi χρησιποιήθηκε η ενέργεια P, όπως δίνεται στο αρχείο Block_LMSc, η οποία υπολογίστηκε μέσω των παρακάτω τύπων:*

$$\mu_i = \frac{\alpha}{P_i} \quad \text{και} \quad P_i(k) = \gamma P_i(k-1) + (1-\gamma)|U_i(k)|^2$$

ΚΑΜΠΥΛΕΣ ΕΚΜΑΘΗΣΗΣ:



Στο παραπάνω σχήμα φαίνεται ότι το τελικό σφάλμα προσεγγίζει πάρα πολύ μικρές τιμές της τάξης του 10^{-25}. Εδώ παρατηρείται διαφορά σε σχέση με την ταχύτητα σύγκλισης σε σχέση με τα δύο προηγούμενα ερωτήματα, καθώς το ανεκτό σφάλμα της τάξης του 10^{-5} επιτυγχάνεται κοντά στο 40 βήμα.

ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ:

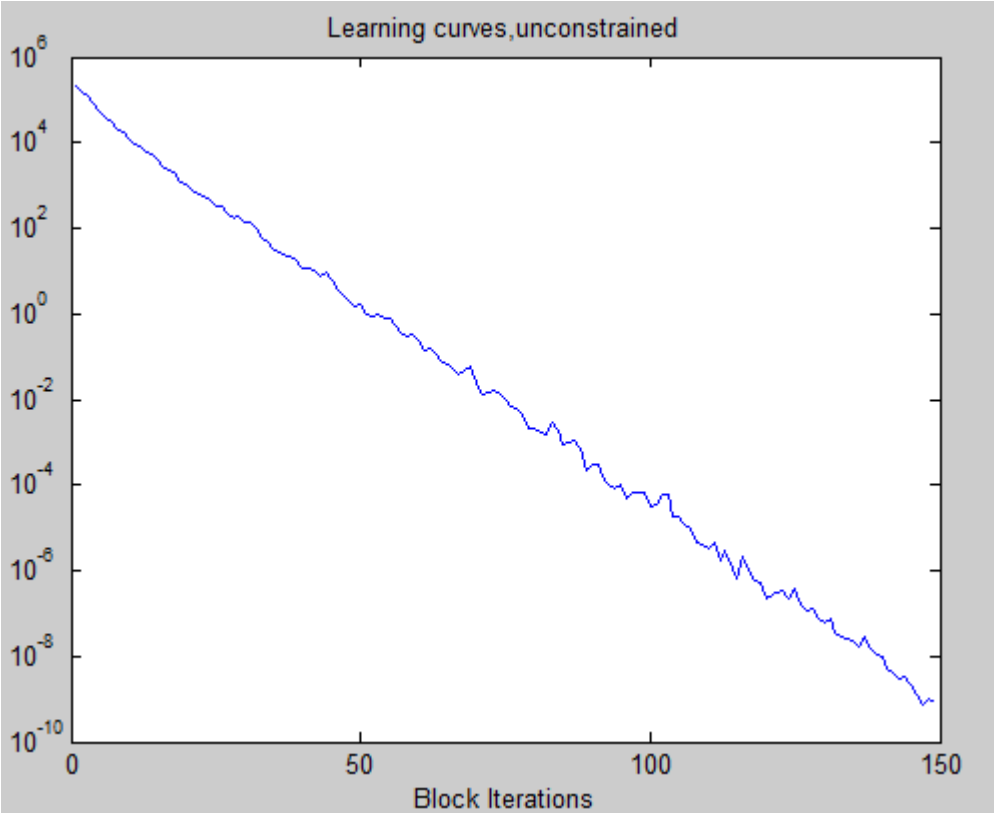
Εισάγοντας τα tic & toc που διαθέτει το matlab για τη μέτρηση χρόνου εκτέλεσης προγραμμάτων βρέθηκε ότι time= 0.4009s, για μήκος εισόδου u 153600, και μήκος φίλτρου 1024 συντελεστές. Παρατηρείται λοιπόν τεράστια διαφορά όσο να αφορά την ταχύτητα σε σχέση με τους προηγούμενους δύο τρόπους . Κάτι τέτοιο είναι αναμενόμενο, αν αναλογιστεί κανείς ο fast block LMS έχει μειωμένη πολυπλοκότητα σε σχέση με τον block LMS.

4) Υλοποίηση με αβίαστη προσαρμογή στο πεδίο της συχνότητας:

ΠΑΡΑΤΗΡΗΣΕΙΣ:

Πρόκειται για παραλλαγή της παραπάνω υλοποίησης η οποία έχει μειωμένη πολυπλοκότητα, καθώς κάνει χρήση μόνο 3 μετασχηματισμών fourier. Επί της ουσίας η υλοποίηση είναι ίδια με προηγουμένως, με μόνη διαφορά ότι έχει αφαιρεθεί πλέον από την υλοποίηση το ορθογώνιο που απεικονίζεται με διακεκομένη γραμμή στο προηγούμενο σχήμα.
Η μετατροπή αυτής της παραλλαγής σε κώδικα βρίσκεται στο αρχείο Block_LMSd.

ΚΑΜΠΥΛΕΣ ΕΚΜΑΘΗΣΗΣ:



Στο παραπάνω σχήμα φαίνεται ότι το τελικό σφάλμα προσεγγίζει πολύ μικρές τιμές της τάξης του 10^{-9}. Εδώ παρατηρείται διαφορά σε σχέση με την ταχύτητα σύγκλισης σε σχέση με το προηγούμενο ερώτημα, καθώς το ανεκτό σφάλμα της τάξης του 10^{-5} επιτυγχάνεται κοντά στο 95 βήμα. Δηλαδή πλέον έχει μειωθεί η ταχύτητα σύγκλισης του αλγορίθμου. Κάτι τέτοιο ήταν αναμενόμενο, καθώς ένα από τα μειονεκτήματα αυτής της παραλλαγής είναι η αύξηση του σφάλματος που παρατηρείται.

ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ:

Τέλος, εισάγοντας τα tic & toc που διαθέτει το matlab για τη μέτρηση χρόνου εκτέλεσης προγραμμάτων βρέθηκε ότι time=0.0654s, για μήκος εισόδου u 153600, και μήκος φίλτρου 1024συντελεστές. Εδώ, ο χρόνος είναι μικρότερος της προηγούμενης υλοποίησης, κάτι που δείχνει και την αποτελεσματικότητα της παραλλαγής που έγινε.