Στατιστική επεξεργασία σήματος και μάθηση

Εργασία 1

Περιεχόμενα

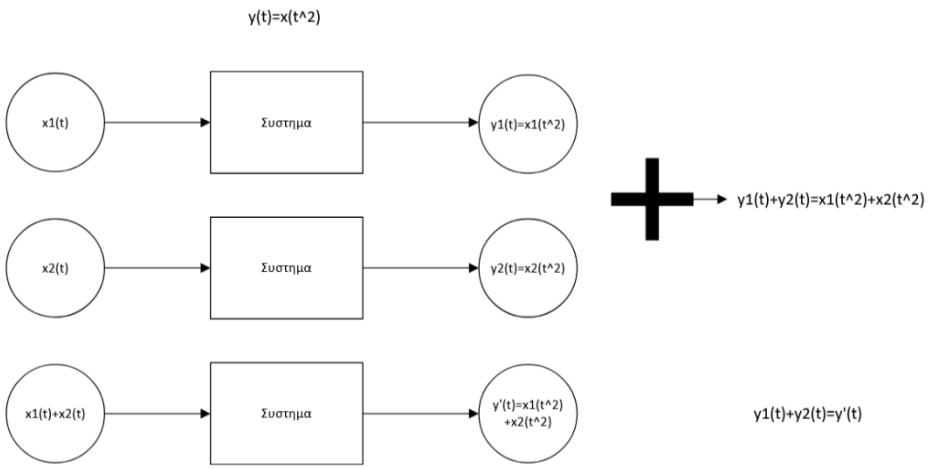
Link για τον κώδικα: https://github.com/GrigorisTzortzakis/Statistical-signal-processing-and-machine-learning

Εισαγωγή

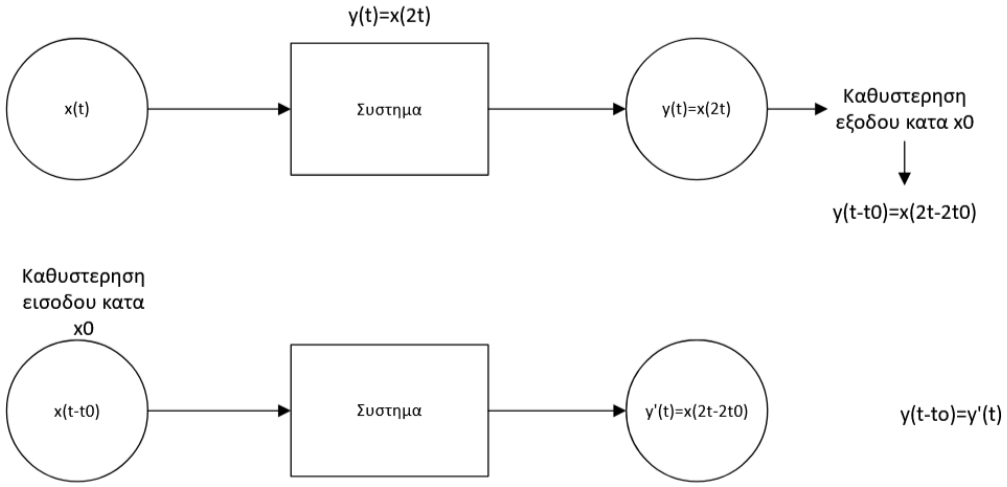
Αρχικά, είναι αναγκαίο να αναλύσουμε μερικές έννοιες. Το πρόβλημα μας περιλαμβάνει ένα ΓΧΑ σύστημα και στην συνέχεια ένα γραμμικά μεταβαλλόμενο σύστημα. Αναλύουμε τις βασικές ιδιότητες του κάθε συστήματος:

ΓΧΑ: Το σύστημα ικανοποιεί τις ιδιότητες γραμμικότητας και είναι χρονικά αμετάβλητο. Αναλύουμε την σημασία αυτών παρακάτω. Επιπλέον, ισχύει ότι η έξοδος του είναι συνέλιξη της εισόδου με την κρουστική απόκριση.

Γραμμικό: Είναι η αρχή της υπέρθεσης, δηλαδή ακολουθεί τους νόμους προσθετικοτητας και ομογένειας. Ο νομός της προσθετικοτητας ορίζει ότι το άθροισμα των εισόδων είναι ίσο με το άθροισμα των εξόδων όταν βάζουμε την κάθε είσοδο ξεχωριστά. Ο νόμος της ομογένειας ορίζει ότι οποιοδήποτε scaling της εισόδου μεταφράζεται σε αντίστοιχο scaling της εξόδου. Για παράδειγμα, ας θεωρήσουμε το σύστημα $y(t) = x(t^2)$.



Χρονικά αμετάβλητο: Οποιαδήποτε χρονική μεταβολή στην είσοδο μεταφράζεται στην ιδιά χρονική μεταβολή στην έξοδο. Για παράδειγμα, ας θεωρήσουμε το σύστημα y(t)=x(2t).



Συμπερασματικά, μπορούμε εύκολα να προβλέψουμε την συμπεριφορά του συστήματος με τις παραπάνω ιδιότητες.

Γραμμικά χρονικά μεταβαλλόμενο: Είναι δυσκολότερο στην πρόβλεψη, καθώς οι χρονικές μεταβολές στην είσοδο δεν μεταφράζονται αντίστοιχα στην έξοδο. Επιπλέον, δεν ισχύει η σχέση της εξόδου με την συνέλιξη εισόδου με την κρουστική απόκριση.

Μερικές ακόμα γενικές παρατηρήσεις πριν προχωρήσουμε στην λύση των ζητουμένων είναι ότι χρησιμοποιείτε ο μετασχηματισμός Z και ότι μας δίνεται η συνάρτηση μεταφοράς ώστε να μπορέσουμε να δούμε αν η πρόβλεψη μας ήταν σωστή και να μοντελοποιησουμε την έξοδο d(n) του αγνώστου συστήματος.

Ζητούμενο 1.1

Αρχικά, στην συγκεκριμένη άσκηση, ο σκοπός του φίλτρου wiener είναι η εύρεση του αγνώστου συστήματος. Θέλουμε δηλαδή να βρούμε την κρουστική του απόκριση παρατηρώντας την έξοδο του συστήματος, η οποία πρακτικά μας περιγράφει την ακριβή λειτουργία του. Γνωρίζουμε ότι η έξοδος του φίλτρου περιγράφεται ως δυο διανύσματα, εκ τον οποίον ένα από αυτά είναι οι συντελεστές του φίλτρου w και το άλλο είναι τα δείγματα της εισόδου x. Τότε, μπορούμε να κατανοήσουμε ότι πρέπει να βρούμε τις τιμές του w καθώς αυτές περιγράφουν τους συντελεστές της H(z). Ο υπολογισμός αυτό γίνεται χρησιμοποιώντας το μέσο τετραγωνικό σφάλμα της εξόδου d(n) του αγνώστου συστήματος και την έξοδο y(n) του φίλτρου. Είναι $\mathrm{MSE} = \mathrm{E}[\mathrm{d}(\mathrm{n}) - \mathrm{y}(\mathrm{n})]^2$.

Συμπερασματικά, το βέλτιστο φίλτρο wiener υπολογίζει τους συντελεστές w ώστε το MSE να ελαχιστοποιείτε.

Ζητούμενο 1.2

Το ερώτημα έχει υλοποιηθεί πλήρως σε matlab και σε python. Σε αυτό το σημείο δίνεται η υλοποίηση matlab και η python μπορεί να βρεθεί στο link στην αρχή της αναφοράς.

```
%  input signal/shma eisodou
N = 1000; % Number of samples/arithmos deigmaton
x = randn(1, N); % White Gaussian noise with 0 mean values and
dispersion of 1/leukos thorivos opos zitite

%  Simulating the Unknown System/prosomiosi tou agnostou sistimatos
h = [1, -0.4, -4, 0.5]; % Coefficients of H(z)/sidelestes pou dinode
d = filter(h, 1, x); % Output signal of the system/eksodon d(n)

% Forming the Input matrix/mhtroo eisodou
L = length(h); % Filter length/mhkos filtrou
X = zeros(N, L);
for i = 1:L
    X(i:end, i) = x(1:end-i+1);
end

%  Computing  Autocorrelation matrix/ipologismos matrhou
```

```
autosisxetisis
R_x = (X' * X) / N;

%  Computing Cross-Correlation Vector/ ipologismos dianismatos
%  autosisxetisis
p = (X' * d') / N;

% Solving the wiener hopf equation/ lisi eksisosis
w = R_x \ p;


disp('Estimated Coefficients of H(z):');
disp(w');
disp('Actual Coefficients of H(z):');
disp(h);
```

Παρατηρούμε ότι εντοπίζει με επιτυχία όλους τους συντελεστές.

```
Estimated Coefficients of H(z):
    1.0000   -0.4000   -4.0000    0.5000

Actual Coefficients of H(z):
    1.0000   -0.4000   -4.0000    0.5000
```

Ζητούμενο 1.3

Ο αλγόριθμος LMS, ομοίως με το φίλτρο wiener, υπολογίζει τους συντελεστές του φίλτρου ώστε να ελαχιστοποιήσει το μέσο τετραγωνικό σφάλμα. Αυτό που αλλάζει είναι ο τρόπος υπολογισμού. Είδαμε ότι το wiener χρησιμοποιεί τις εξισώσεις wiener-hopf, δηλαδή λύνει ένα γραμμικό σύστημα. Ο lms αντιθέτως είναι επαναληπτική διαδικασία (iterative αλγόριθμος) και λειτουργεί σε πραγματικό χρόνο, χωρίς να χρειάζεται ανάλυση στατιστικής του σήματος.

Συγκεκριμένα, ο LMS υπολογίζει τους συντελεστές με τον εξής τρόπο:

$w(n+1) = w(n) + \mu * e(n) * x(n)$ όπου το w(n) είναι το διάνυσμα των συντελεστών, το μ είναι το μέγεθος του βήματος, το x(n) είναι το διάνυσμα εισόδου και το e(n) είναι το σφάλμα (η διαφορά) ανάμεσα στο επιθυμητό σήμα εξόδου d(n) (του αγνώστου συστήματος) και στο σήμα εξόδου του φίλτρου. Οι αρχικές τιμές πρόβλεψης για τα βάρη είναι 0.

Προχωρώντας, ορίζουμε το διάνυσμα σφάλματος $w'(n) = w(n) - wo$, το οποίο μας λέει πόσο απέχουν οι συντελεστές μας w(n) από τους βέλτιστους wo. Αντικαθιστώντας το στην εξίσωση περνούμε $w'(n+1) = w'(n) - \mu * x(n) * x^{T}(n) * w'(n) + \mu * x(n) * e0(n)$, οπού γνωρίζουμε ότι $e0(n) = d(n) - w0^{T} * x(n)$.

Στην συνέχεια, γράφουμε την εξίσωση ως $E[w'(n+1)] = (I - \mu * Rx) * E[w'(n)]$. Παρατηρούμε ότι το μέσο διάνυσμα σφάλματος βάρους γίνεται scale από το μητρώο I-μ*Rx, άρα για να συγκλίνει (δηλαδή να προσεγγίζει το 0) πρέπει οι ιδιωτίες του μητρώου αυτού να βρίσκονται εντός του μοναδιαιου κύκλου. Τότε είναι $0 < \mu < \frac{2}{\lambda max}$, όπου το λmax είναι η μέγιστη ιδιοτιμη του μητρώου αυτοσυσχετισης Rx.

Τέλος, αν αντικαταστήσουμε στην προηγουμένη εξίσωση E[w'(n+1)] τον όρο για το μέσο τετραγωνικό σφάλμα (δηλαδή απλά να τετραγωνίσουμε αυτό τον όρο), βλέπουμε ότι το άνω σφάλμα εξαρτάται από το μ και από το σήμα εισόδου.

Ζητούμενο 1.3.1

Στο προηγούμενο ερώτημα είδαμε ότι για να συγκλίνει ο αλγόριθμος, πρέπει οι ιδιωτίες του μητρώου I-μ*R να βρίσκονται εντός του μοναδιαιου κύκλου ώστε να μειώνεται το E[w'(n)] με κάθε επανάληψη. Άρα χρειαζόμαστε την συνθήκη $|1 - \mu * \lambda i| < 1$. Επεκτείνοντας την ανισότητα έχουμε $-1 < 1 - \mu * \lambda i < 1$ και λυνοντας την τελικά έχουμε $0 < \mu < \frac{2}{\lambda max}$.

Ζητούμενο 1.3.2

```
N = 1000; % Number of samples/arithmos deigmaton
x = randn(1, N); % Input signal/sima eisodou
h = [1, -0.4, -4, 0.5]; % Given coefficients/H(z)
d = filter(h, 1, x); % d(n)

L = 4; % 4 coefficients/4 sidelestes
w = zeros(1, L); % Initial coefficients of lms are zero/miden arxikoi
sidelestes

% Computing the autocorrelation matrix Rx/Ipologismos mitroou Rx
X_full = zeros(L, N-L+1);
for i = 1:L
    X_full(i, :) = x(i:N-L+i);
end
R_x = (X_full * X_full') / (N - L);

% Maximum step size/Prosdiorismos μ
lambda_max = max(eig(R_x)); % Largest eigenvalue of R_x
mu_max = 2 / lambda_max;
mu = 0.1 * mu_max;

% Lms implementation/ilopoihsh lms
for n = L:N

    X = [x(n), x(n-1), x(n-2), x(n-3)];
```

```
    % filter output/eksodos filtrou
    y = w * X';

    % Error of filter output signal compared to the desired
signal/sfalma
    % eksodou
    e = d(n) - y;

    % New filter coefficients/Enimerosi sideleston me kathe epanalipsi
    w = w + mu * e * X;
end


disp('Estimated LMS Filter Coefficients/Lms apotelesmata:');
disp(w);
disp('Actual System Coefficients/Pragmatika apotelesmata:');
disp(h);
```

Συγκρίνοντας το με τα αποτελέσματα του 1.2, βλέπουμε ότι είναι ακριβώς ίδια, δηλαδή εντόπισε με επιτυχία τους σωστούς συντελεστές. Αυτό οφείλετε στην σωστή επιλογή του μ που επιτρέπει στον αλγόριθμο να συγκλίνει και να βρίσκεται εντός του ορίου που θέτει το άνω φράγμα.

```
Estimated LMS Filter Coefficients/Lms apotelesmata:
    1.0000   -0.4000   -4.0000    0.5000

Actual System Coefficients/Pragmatika apotelesmata:
    1.0000   -0.4000   -4.0000    0.5000
```

Ζητούμενο 1.3.3

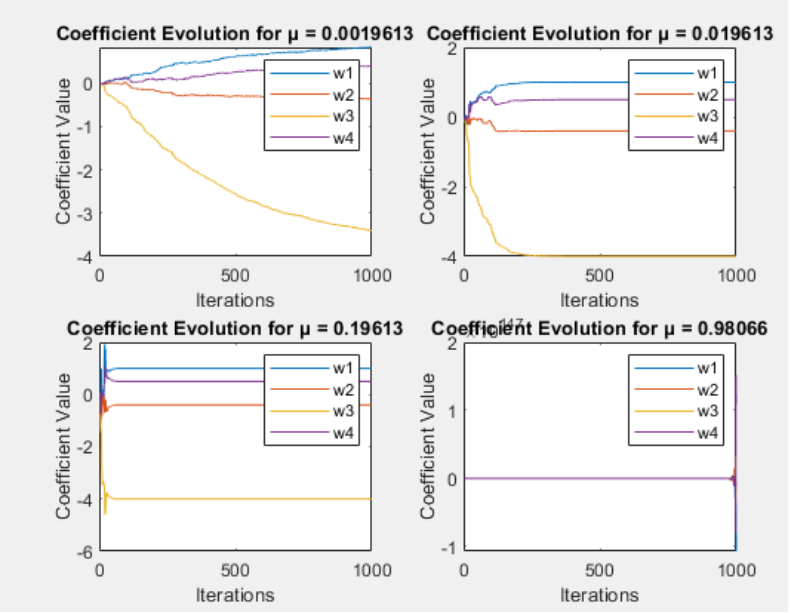
```
N = 1000; % Samples/deigmata
x = randn(1, N); % Input signal/sima eisodou
h = [1, -0.4, -4, 0.5]; % Coefficients/Sidelestes
d = filter(h, 1, x); % d(n)

L = 4; % 4 coefficients/4 sidelestes

% Computing the autocorrelation matrix Rx/Ipologismos Rx
X_full = zeros(L, N - L + 1);
for i = 1:L
    X_full(i, :) = x(i:N - L + i);
end
```

```
R_x = (X_full * X_full') / (N - L);

% Maximum step size μ/ Megisti timh tou μ
lambda_max = max(eig(R_x));
mu_max = 2 / lambda_max;

% Step sizes required/thetoume to μ pou zitite
mu_values = [0.001, 0.01, 0.1, 0.5] * mu_max;


coeff_evolution = zeros(length(mu_values), N, L);

% Loop for all step sizes/kanoume loop oste na treksoun ola ta μ
final_coefficients = zeros(length(mu_values), L);
for i = 1:length(mu_values)
    mu = mu_values(i);
    w = zeros(1, L);


    coeffs_over_time = zeros(N, L);

    % Lms implementation/ilopoihsh lms
    for n = L:N

        X = [x(n), x(n-1), x(n-2), x(n-3)];

       % filter output/eksodos filtrou
        y = w * X';

         % Error of filter output signal compared to the desired
signal/sfalma
    % eksodou
        e = d(n) - y;

       % New filter coefficients/Enimerosi sideleston me kathe
epanalipsi
        w = w + mu * e * X;


        coeffs_over_time(n, :) = w;
    end

    % Save coefficient evolution/ekseliksei sideleston
    coeff_evolution(i, :, :) = coeffs_over_time;

    % Save coefficients/sidelestes
    final_coefficients(i, :) = w;
end
```

```
for i = 1:length(mu_values)
    mu = mu_values(i);
    disp(['Step size μ: ', num2str(mu)]);
    disp('Final Coefficients:');
    disp(final_coefficients(i, :));
    disp('-------------------------------------');
end


figure;
for i = 1:length(mu_values)
    subplot(2, 2, i);
    plot(squeeze(coeff_evolution(i, :, :)));
    title(['Coefficient Evolution for μ = ', num2str(mu_values(i))]);
    xlabel('Iterations');
    ylabel('Coefficient Value');
    legend('w1', 'w2', 'w3', 'w4');
end
```

Παρατηρούμε ότι για τις οριακές τιμές του μ (0.001, 0.5 μmax) ο αλγόριθμος δεν συγκλίνει. Μάλιστα, για την τιμή 0.5 ο αλγόριθμος ξεπερνά το άνω φράγμα και συνεπώς τα αποτελέσματα του είναι εντελώς λάθος, αφού ουσιαστικά όχι μόνο δεν ολοκληρώνεται, αλλά ταυτόχρονα δεν είναι σταθερός. Για την τιμή 0.01 παρατηρούμε ότι συγκλίνει με πιο αργό ρυθμό από την 0.1. Συμπερασματικά, κατανοούμε ότι οφείλουμε να βρούμε το κατάλληλο μ, καθώς αν είναι πολύ μικρό ο αλγόριθμος συγκλίνει πολύ αργά, ενώ αν είναι μεγάλο ο αλγόριθμος αποτυγχάνει εντελώς.

```
Step size μ: 0.0019613
Final Coefficients:
    0.8315   -0.3601   -3.4008    0.3932

-------------------------------------
Step size μ: 0.019613
Final Coefficients:
    1.0000   -0.4000   -4.0000    0.5000

-------------------------------------
Step size μ: 0.19613
Final Coefficients:
    1.0000   -0.4000   -4.0000    0.5000

-------------------------------------
Step size μ: 0.98066
Final Coefficients:
  1.0e+147 *

   -1.0687   -0.0608    0.4831    1.0980
```



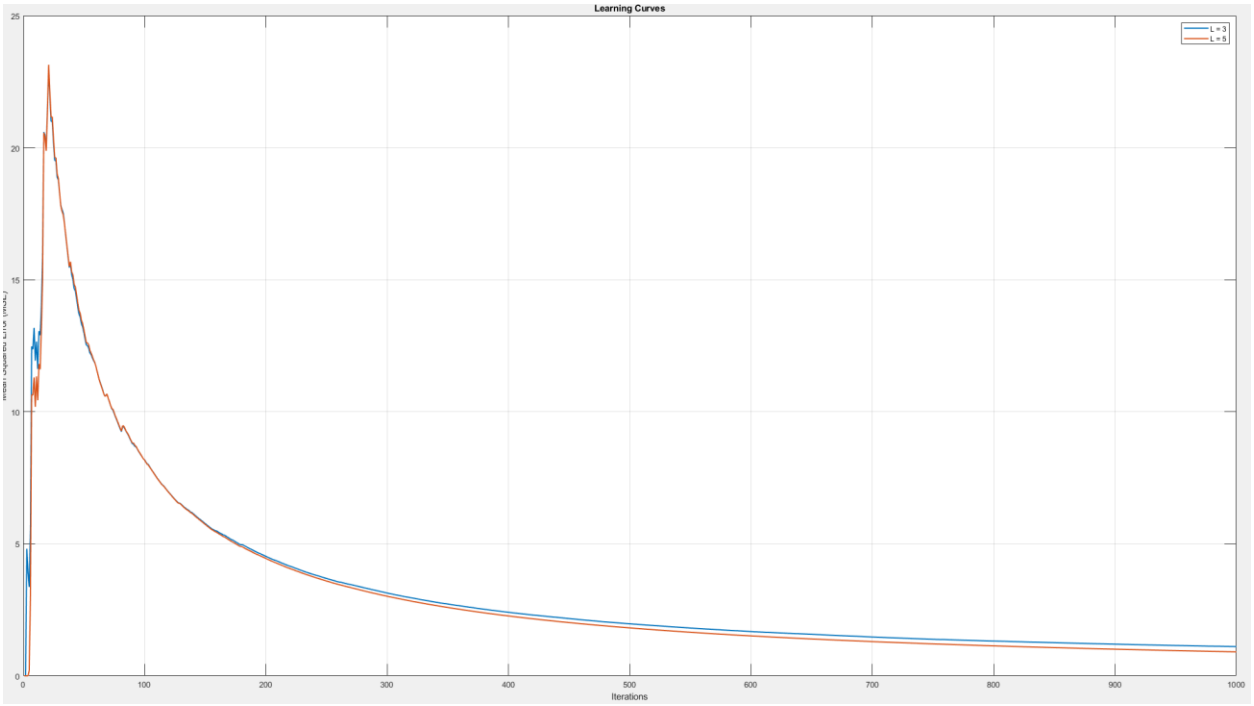
Ζητούμενο 1.3.3.1

```
N = 1000; % Samples/deigmata
x = randn(1, N); % Input signal/sima eisodou
h = [1, -0.4, -4, 0.5]; % Coefficients/Sidelestes
d = filter(h, 1, x); % d(n)
mu = 0.01; % Step size/vima μ

% Filter length/mhkos filtrou
filter_lengths = [3, 5];


learning_curves = zeros(length(filter_lengths), N);

% Loop for each length/epanalipseis gia to kathe mhkos
for idx = 1:length(filter_lengths)
    L = filter_lengths(idx);
    w = zeros(1, L);
    mse = zeros(1, N);

    % Lms implementation/ilopoihsh lms
    for n = L:N

        X = x(n:-1:n-L+1);

        % filter output/eksodos filtrou
        y = w * X';

       % Error of filter output signal compared to the desired
signal/sfalma
    % eksodou
        e = d(n) - y;

        %  New filter coefficients/Enimerosi sideleston me kathe
epanalipsi
        w = w + mu * e * X;


        mse(n) = e^2;
    end


    learning_curves(idx, :) = cumsum(mse) ./ (1:N);
end


figure;
for idx = 1:length(filter_lengths)
```

```
    plot(1:N, learning_curves(idx, :), 'LineWidth', 1.5); hold on;
end
title('Learning Curves');
xlabel('Iterations');
ylabel('Mean Squared Error (MSE)');
legend(arrayfun(@(L) sprintf('L = %d', L), filter_lengths,
'UniformOutput', false));
grid on;
```

Η καμπύλη μάθησης του αλγορίθμου είναι το μέσο τετραγωνικό σφάλμα.

Παρατηρούμε ότι με μικρότερο πλήθος συντελεστών ο αλγόριθμος συγκλίνει πιο γρήγορα αλλά έχει μεγαλύτερο Mse επειδή δεν μπορεί να προβλέψει με ακρίβεια το άγνωστο σύστημα. Με μεγαλύτερο πλήθος συντελεστών ο αλγόριθμος έχει μικρότερο Mse αλλά συγκλίνει πιο αργά.



Ζητούμενο 2.3

Στην εισαγωγή αναλύσαμε την σημασία των ΓΧΑ συστημάτων. Ένα γραμμικά χρονικά μεταβαλλόμενο σύστημα σημαίνει ότι η συμπεριφορά του συστήματος αλλάζει με την πάροδο του χρόνου, άρα το πρόβλημα εύρεσης του αγνώστου συστήματος γίνεται αυτόματος πιο δύσκολο.

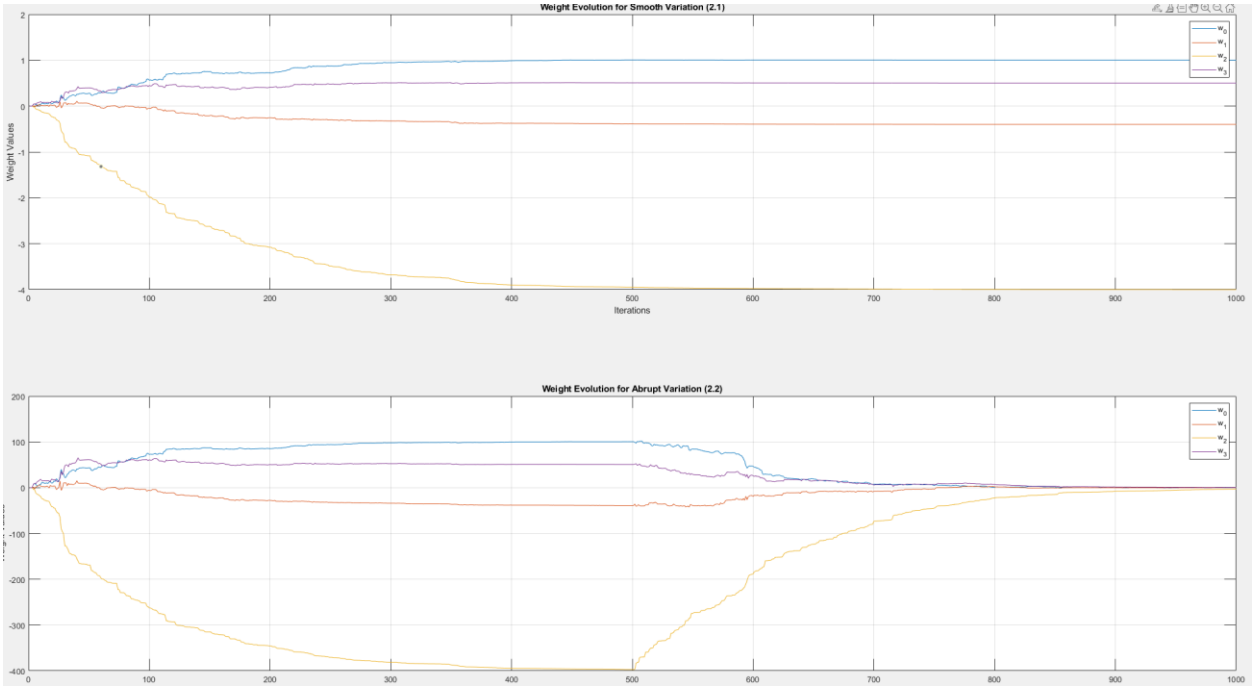
```
N = 1000; % Samples/deigmata
x = randn(1, N); % Input signal/sima eisodou
L = 4; % 4 coefficients/4 sidelestes
mu = 0.01; % Step size


mse_smooth = zeros(1, N);
mse_abrupt = zeros(1, N);


weights_smooth = zeros(N, L);
weights_abrupt = zeros(N, L);
% Smooth Variation/Omalh metavolh
b_smooth = 1 ./ (1 + exp(-0.02 * (1:N))); % b(n)
h_smooth = zeros(L, N);
for n = 1:N
    h_smooth(:, n) = b_smooth(n) * [1; -0.4; -4; 0.5];
end

% d(n)
d_smooth = zeros(1, N);
for n = L:N
    d_smooth(n) = h_smooth(:, n)' * flip(x(n-L+1:n))';
end

% Lms implementation/ilopoihsh lms
w = zeros(1, L);
for n = L:N
    X = x(n:-1:n-L+1);
    y = w * X'; % filter output/eksodos filtrou
    e = d_smooth(n) - y; % Error of filter output signal compared to
the desired signal/sfalma eksodou
    w = w + mu * e * X; % New filter coefficients/Enimerosi sideleston
me kathe epanalipsi
    weights_smooth(n, :) = w;
    mse_smooth(n) = e^2;
end

% Abrupt Variation/akariaia metavolh
b_abrupt = [ones(1, 500) * 100, zeros(1, 500)]; % b(n)
h_abrupt = zeros(L, N);
for n = 1:N
    h_abrupt(:, n) = b_abrupt(n) * [1; -0.4; -4; 0.5];
end

% d(n)
d_abrupt = zeros(1, N);
for n = L:N
```

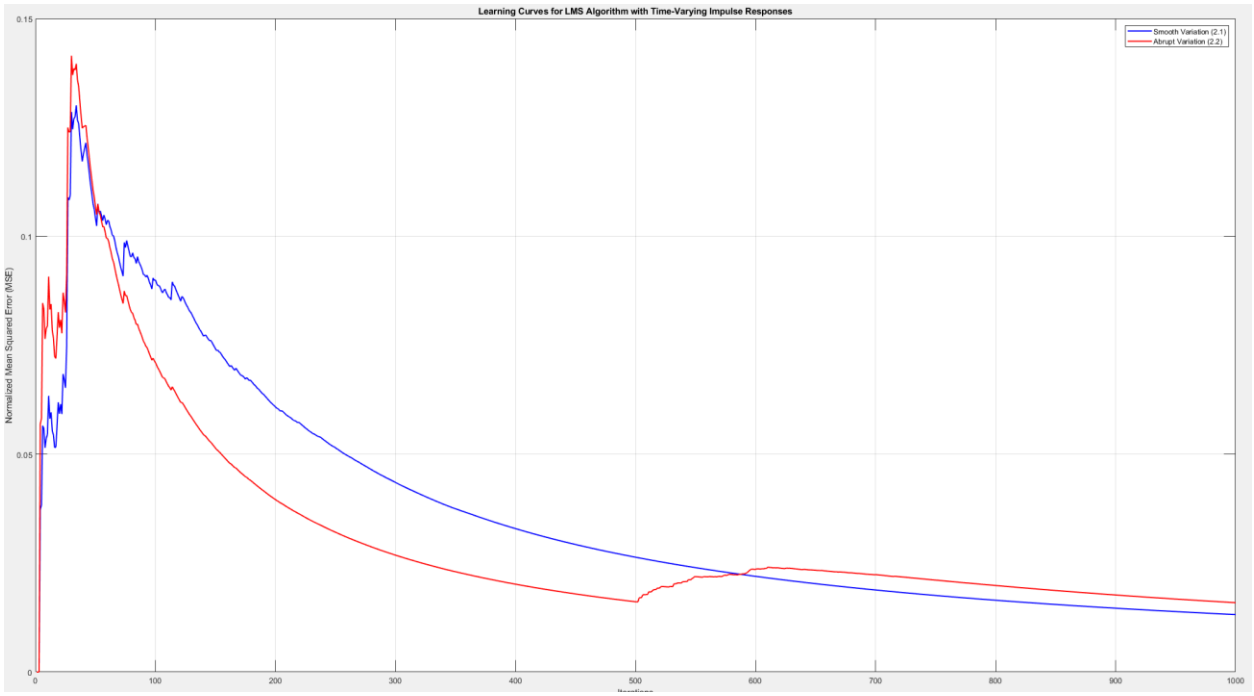
```
    d_abrupt(n) = h_abrupt(:, n)' * flip(x(n-L+1:n))';
end

% Lms implementation/ilopoihsh lms
w = zeros(1, L);
for n = L:N
    X = x(n:-1:n-L+1);
    y = w * X'; % filter output/eksodos filtrou
    e = d_abrupt(n) - y; % Error of filter output signal compared to
the desired signal/sfalma eksodou
    w = w + mu * e * X; % New filter coefficients/Enimerosi sideleston
me kathe epanalipsi
    weights_abrupt(n, :) = w;
    mse_abrupt(n) = e^2;
end


mse_smooth = mse_smooth / max(mse_smooth);
mse_abrupt = mse_abrupt / max(mse_abrupt);


figure;
plot(1:N, cumsum(mse_smooth) ./ (1:N), 'b', 'LineWidth', 1.5); hold
on;
plot(1:N, cumsum(mse_abrupt) ./ (1:N), 'r', 'LineWidth', 1.5);
title('Learning Curves for LMS Algorithm with Time-Varying Impulse
Responses');
xlabel('Iterations');
ylabel('Normalized Mean Squared Error (MSE)');
legend('Smooth Variation (2.1)', 'Abrupt Variation (2.2)');
grid on;


figure;
subplot(2, 1, 1);
plot(weights_smooth);
title('Weight Evolution for Smooth Variation (2.1)');
xlabel('Iterations');
ylabel('Weight Values');
legend('w_0', 'w_1', 'w_2', 'w_3');
grid on;

subplot(2, 1, 2);
plot(weights_abrupt);
title('Weight Evolution for Abrupt Variation (2.2)');
xlabel('Iterations');
ylabel('Weight Values');
legend('w_0', 'w_1', 'w_2', 'w_3');
```

```
grid on;


disp('Final Weights for Smooth Variation:');
disp(weights_smooth(end, :));

disp('Final Weights for Abrupt Variation:');
disp(weights_abrupt(end, :));
```

Από τα αποτελέσματα διακρίνουμε ότι έως τις 500 επαναλήψεις η ακαριαία μεταβολή συγκλίνει πιο γρήγορα, όμως μετρά αυξάνεται το Mse και επιπλέον οι συντελεστές αλλάζουν εντελώς τιμή. Το αποτέλεσμα είναι λογικό, στην ομαλή μεταβολή ο αλγόριθμος μπορεί με ακρίβεια να εντοπίσει τους σωστούς συντελεστές και να συγκλίνει με επιτυχία, ενώ στην ακαριαία πρακτικά δεν καταφέρνει να εντοπίσει τους συντελεστές μετα από την αλλαγή του συντελεστή.

```
Final Weights for Smooth Variation:
    1.0002   -0.3999   -3.9996    0.5001

Final Weights for Abrupt Variation:
   -0.1604    0.1742   -2.8665    0.6833
```



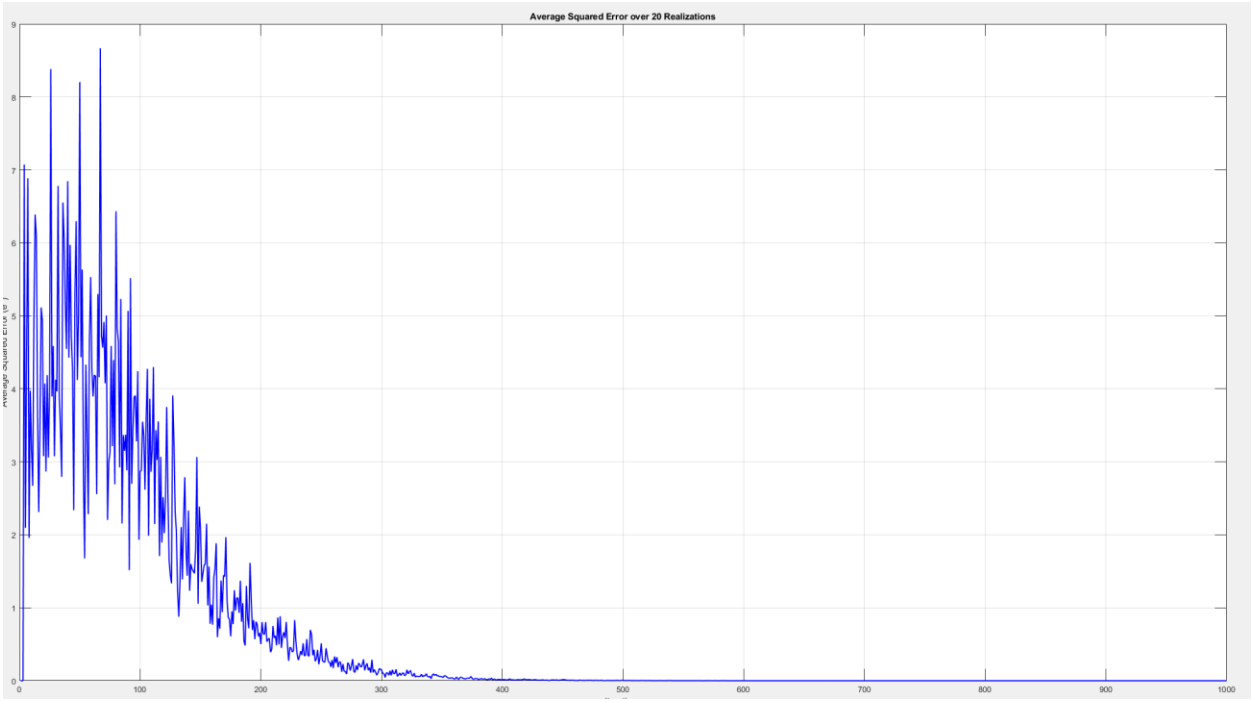


Ζητούμενο 2.4

```
N = 1000; % Samples/deigmata
L = 4; % 4 coefficients/4 sidelestes
mu = 0.01; % Step size
num_realizations = 20; % 20 different inputs/20 ilopoihseis


all_squared_errors = zeros(num_realizations, N);

% Lms implementation/ilopoihsh lms
for realization = 1:num_realizations
    % New Input signal each time for 20 times/Neo sima eisodou 20
fores
    x = randn(1, N);

    % Smooth Variation/Omalh metavolh
    b_smooth = 1 ./ (1 + exp(-0.02 * (1:N))); % b(n)
    h_smooth = zeros(L, N);
    for n = 1:N
        h_smooth(:, n) = b_smooth(n) * [1; -0.4; -4; 0.5];
    end

    %  d(n)
```

```
    d_smooth = zeros(1, N);
    for n = L:N
        d_smooth(n) = h_smooth(:, n)' * flip(x(n-L+1:n))';
    end


    w = zeros(1, L);
    squared_error = zeros(1, N);
    for n = L:N
        X = x(n:-1:n-L+1);
        y = w * X'; % filter output/eksodos filtrou
        e = d_smooth(n) - y; % Error of filter output signal compared
to the desired signal/sfalma eksodou
        w = w + mu * e * X; % New filter coefficients/Enimerosi
sideleston me kathe epanalipsi
        squared_error(n) = e^2;
    end


    all_squared_errors(realization, :) = squared_error;
end

% Average squared error/mesos oros sfalmatos
avg_squared_error = mean(all_squared_errors, 1);


figure;
plot(1:N, avg_squared_error, 'b', 'LineWidth', 1.5);
title('Average Squared Error over 20 Realizations');
xlabel('Iterations');
ylabel('Average Squared Error (e^2)');
grid on;
```

Παρατηρούμε ότι αρχικά το Mse είναι υψηλό, το οποίο συμβαίνει επειδή δεν γνωρίζουμε αρχικά τους συντελεστές και είναι 0. Μετα από περίπου 400 επαναλήψεις, ο αλγόριθμος συγκλίνει αφού βλέπουμε ότι το σφάλμα πρακτικά ελαχιστοποιείτε, συνεπώς εντοπίσαμε με επιτυχία τους συντελεστές. Το μήκος βήματος επίσης είναι εντός του άνω φράγματος, αφού το Mse παραμένει σταθερό.


Average Squared Error over 20 Realizations
0
1
2
3
4
5
6
7
8
9
0
100
200
300
400
500
600
700
800
900
1000