Code

Faccin Dario, Pegoraro Jacopo

1 Generation of the signal

```
close all; clear global; clearvars; clc;

%% GENERATE THE PROCESS x(k), 1 REALIZATION

Nsamples=800;
% Frequencies of the exponentials
f1=0.17;
f2=0.78;
% Generate the white noise (2 components)
%choose the correct variance
%sigmaw=2;
sigmaw=0.1;
% Real part
wi=sigmaw*randn(Nsamples,1);
% Imaginary part
wq=sigmaw*randn(Nsamples,1);
% Generate the initial phases
phi1=2*pi*rand(1);
phi2=2*pi*rand(1);

xi=zeros(Nsamples,1);
xq=zeros(Nsamples,1);
for k=1:Nsamples
xi(k)=cos(2*pi*f1*k+phi1)+0.8*cos(2*pi*f2*k+phi2)+wi(k);
xq(k)=sin(2*pi*f1*k+phi1)+0.8*sin(2*pi*f2*k+phi2)+wq(k);
end

% Complex r.p. x(k), 800 samples
x=xi+1i*xq;

save('inputsignal01.mat', 'x');
```

2 Exercise 1

```
%% LOAD 1 REALIZATION OF THE PROCESS

load('inputsignal2.mat', 'x');
```

```
Nsamples=length(x);

%% SPECTRAL ANALYSIS

% Autocorrelation (unbiased estimate)
[rx_full]=autocorrelation_Unb(x);
L=floor(Nsamples/2);%L should be lower than the length of the r.p. because of the
% high variance when n approaches K
rx=rx_full(1:L);

% Blackman-Tukey correlogram
% The length of 2*L+1 is because of page 86 note 24
w_bt=window(@hamming,2*L+1);
Pbt1=correlogram(x, w_bt, rx, L);
%Pbt2=correlogram(x, w_rect, rx, L);

% Periodogram
X=fft(x);
Pper=(1/Nsamples)*(abs(X)).^2;

% Welch periodogram
S=60;   %overlap
D=120;   %window length
%w_welch=window(@hamming,D);
w_welch=window(@rectwin,D);
%w_welch=kaiser(D,5);
[Welch_P, Ns] = welchPSD(x, w_welch, S);
var_Welch=Welch_P.^2/Ns;


% Analytical PSD: compute the transform of rx(n) on paper and plot it
% according to the requirements
%sigmaw=0.1
b = zeros(1,800);
for i=1:length(b)
%change to 0.1 for es2
b(i) = 10*log10(2);
end

b(ceil(0.17*800)) = 10*log10(Nsamples);
b(ceil(0.78*800)) = 0.8*10*log10(Nsamples);

%% Choice of N
N = 25;

[copt, Jmin]=predictor(rx, N);
t=40;
Jvect=zeros(t,1);

for i=1:length(Jvect)
```

```
[c_it, J_it]=predictor(rx, i);
Jvect(i)=J_it;
end

figure('Name', 'J over N');
plot(1:t,10*log10(Jvect));
title('J_{min} over N');
xlim([1 t]);
xlabel('N'); ylabel('J_{min} [dB]');
% coeff=[1; copt];
% A = tf([1 copt.'], 1,1);
% figure, pzmap(A)

%% AR model
% Coefficients of Wiener filter
[a, s_white, d]=findAR(N, rx);
[H_w, omega] = freqz(1, [1; a], Nsamples, 'whole');

%% Final spectral plot
figure('Name', 'Spectral Analysis');
hold on;
plot((1:Nsamples)/Nsamples, 10*log10(Welch_P), 'r')
plot((1:Nsamples)/Nsamples, 10*log10(abs(Pbt1)), 'b')
plot((1:Nsamples)/Nsamples, 10*log10(Pper), 'g:')
plot(omega/(2*pi), 10*log10(s_white*(abs(H_w)).^2), 'Color', 'm');
plot((1:Nsamples)/Nsamples, b, 'k:');
title('Spectral analysis');
legend('Welch', 'Correlogram', 'Periodogram', ['AR(' int2str(N) ')'], 'Actual value',
 'Location', 'SouthWest');
hold off;
xlabel('Normalized frequency');
ylabel('Estimated PSD (dB)');
ylim([-15 30]);
```

3 Exercise 2-3

```
clc; close all; clear global; clearvars;

%% LOAD 1 REALIZATION OF THE PROCESS

load('inputsignal01.mat', 'x');

Nsamples=length(x);

%% SPECTRAL ANALYSIS

% Autocorrelation (unbiased estimate)
[rx_full]=autocorrelation_Unb(x);
```

```
L=floor(Nsamples/2);%L should be lower than the length of the r.p. because of the high
% variance when n approaches K
rx=rx_full(1:L);

% Blackman-Tukey correlogram
% The length of 2*L+1 is because of page 86 note 24
%w_rect=window(@rectwin,2*L+1);
w_bt=window(@blackmanharris,2*L+1);
Pbt1=correlogram(x, w_bt, rx, L);
%Pbt2=correlogram(x, w_rect, rx, L);

% Periodogram
X=fft(x);
Pper=(1/Nsamples)*(abs(X)).^2;

% Welch periodogram
S=80;   %overlap
D=160;   %window length
w_welch=window(@hamming,D);
%w_welch=window(@rectwin,D);
%w_welch=kaiser(D,5);
[Welch_P, Ns] = welchPSD(x, w_welch, S);
var_Welch=Welch_P.^2/Ns;


% Analytical PSD: compute the transform of rx(n) on paper and plot it
% according to the requirements
%sigmaw=0.1
b = zeros(1,800);
for i=1:length(b)
%change to 0.1 for es2
b(i) = 10*log10(0.1);
end


b(ceil(0.17*800)) = 10*log10(Nsamples);
b(ceil(0.78*800)) = 0.8*10*log10(Nsamples);

%% Choice of N
N = 2;

[copt, Jmin]=predictor(rx, N);
t=20;
Jvect=zeros(t,1);

for i=1:length(Jvect)
[c_it, J_it]=predictor(rx, i);
Jvect(i)=J_it;
end
```

```
figure('Name', 'J over N');
plot(1:t,10*log10(Jvect));
title('J_{min} over N');
xlim([1 t]);
xlabel('N'); ylabel('J_{min} [dB]');
% coeff=[1; copt];
% A = tf([1 copt.'], 1,1);
% figure, pzmap(A)

%% AR model
% Coefficients of Wiener filter
[a, s_white, d]=findAR(N, rx);
[H_w, omega] = freqz(1, [1; a], Nsamples, 'whole');

%% Final spectral plot
figure('Name', 'Spectral Analysis');
hold on;
plot((1:Nsamples)/Nsamples, 10*log10(Welch_P), 'r')
plot((1:Nsamples)/Nsamples, 10*log10(abs(Pbt1)), 'Color', 'b')
plot((1:Nsamples)/Nsamples, 10*log10(Pper), 'g:')
plot(omega/(2*pi), 10*log10(s_white*(abs(H_w)).^2), 'Color', 'm');
plot((1:Nsamples)/Nsamples, b, 'k:');
title('Spectral analysis');
legend('Welch', 'Correlogram', 'Periodogram', ['AR(' int2str(N) ')'], 'Actual value',
 'Location', 'SouthWest');
hold off;
xlabel('Normalized frequency');
ylabel('Estimated PSD (dB)');
ylim([-30 30]);

[H, www] = freqz([1; a], 1, Nsamples, 'whole');
figure('Name', 'Z-plane for error predictor A(z)');
zplane([1; a].', 1);
title('Z-plane for error predictor A(z)');

%% Estimation of phases phi1 and phi2 based on coefficients of A(z)

% We notice that we have poles close to the unit circle at phases
% corresponding to phi2 and phi2
ph = angle(roots([1;a]))/(2*pi);
```

4 Exercise 4

```
clc; close all; clear global; clearvars;

%% Least Mean Squares Estimation

% Load one realization
load('inputsignal01.mat','x');
```

```
% Set parameters
L = floor(length(x)/5);
N = 2;
rx = autocorrelation_Unb(x);
rx = rx(1:L);
[a, s_white] = findAR(N, rx);
K = L;

% Max number of iterations
max_iter = 800;

% Coefficients and error initialization
c = zeros(N, max_iter + 1);
e = zeros(1, max_iter);

% Choice of parameter mu
mu_tilde = 0.05;
mu = mu_tilde/(rx(1)*N);

% Center signal around its mean
z = x - mean(x);

for k = 1:max_iter
if (k < N + 1)
% Input vector of length N
x_in = flipud([zeros(N - k + 1, 1); z(1:k - 1)]);
% For k = 1 z(1:0) is an empty matrix
y_k = x_in.'*c(:, k);
else
% Revert vector to obtain values from k-1 to k-N
x_in = flipud(z((k - N):(k-1)));
y_k = x_in.'*c(:, k);
end
% d(k) is input signal z(k)
e_k = z(k) - y_k;
e(k) = e_k;
% Update step
c(:, k+1) = c(:, k) + mu*e_k*conj(x_in);
end

load('Jmin.mat', 'mean_error');
load('avg_coeff.mat', 'c_mean');

for index = 1:N
figure('Name', ['Coefficient of index ' int2str(index)]);
subplot(2, 1, 1)
plot(1:max_iter+1, real(c(index, :)));
hold on;
plot([1, max_iter+1], -real(a(index))*[1 1]);
```

```
hold on;
plot(1:max_iter+1, real(c_mean(index, :)));
title(['Real part of c' int2str(index) ' and c_{opt}' int2str(index)]);
legend(['c' int2str(index)], ['c_{est}' int2str(index)], ['average' int2str(index)]);
xlim([0 max_iter]);



subplot(2, 1, 2);
plot(1:max_iter+1, imag(c(index, :)));
hold on;
plot([1,max_iter+1], -imag(a(index))*[1 1]);
hold on;
plot(1:max_iter+1, imag(c_mean(index, :)));
title(['Imaginary part of c' int2str(index) ' and c_{opt}' int2str(index)]);
legend(['c' int2str(index)], ['c_{est}' int2str(index)],['average' int2str(index)]);
xlim([0 max_iter]);
xlabel('Number of iterations')
end

figure('Name', 'Error function');
plot(1:max_iter, 10*log10(abs(e).^2), 1:max_iter, 10*log10(s_white)*ones(1, max_iter),
 1:max_iter, 10*log10(mean_error));

title('Error function at each iteration');
legend('|e(k)|^2', 'J_{min}','J(k)');
ylim([-15 10])
xlabel('k')
ylabel('|e(k)|^2, J_{min}, J(k)')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc; close all; clear global; clearvars;

%% Least Mean Square, mean over 300 realizations

% Load one realization
load('realizations.mat','x');

% Max number of iterations
max_iter = 800;
% Set parameters
L = floor(size(x,1)/5);
N = 2;
K = L;

% Error vector initialization
e = zeros(size(x,2), max_iter);
C1=zeros(300,801);
C2=zeros(300,801);
```

```
for i=1:size(x,2)
% Autocorrelation
rx = autocorrelation_Unb(x(:,i));
rx = rx(1:L);
[a, s_white] = findAR(N, rx);
% Coefficients initialization
c = zeros(N, max_iter + 1);
% Choice of parameter mu
mu_tilde = 0.05;
mu = mu_tilde/(rx(1)*N);

% Center signal around its mean
z = x(:,i) - mean(x(:,i));

for k = 1:max_iter
if (k < N + 1)
% Input vector of length N
x_in = flipud([zeros(N - k + 1, 1); z(1:k - 1)]);
% For k = 1 z(1:0) is an empty matrix
y_k = x_in.'*c(:, k);
else
% Revert vector to obtain values from k-1 to k-N
x_in = flipud(z((k - N):(k-1)));
y_k = x_in.'*c(:, k);
end
% d(k) is input signal z(k)
e_k = z(k) - y_k;
e(i,k) = e_k;
% Update step
c(:, k+1) = c(:, k) + mu*e_k*conj(x_in);
end
c1=c(1,:);
c2=c(2,:);
C1(i,:)=c1;
C2(i,:)=c2;
end

c_mean(1,:)=mean(C1);
c_mean(2,:)=mean(C2);
% Mean error is computed over 300 errors for the same k

%mean_error = zeros(1, max_iter);
mean_error = mean(abs(e.^2));
%for k=1:max_iter
%mean_error(k) = sum(e(:,k))/size(e,1);
%end


for index = 1:N
figure('Name', ['Coefficient of index ' int2str(index)]);
```

```
subplot(2, 1, 1)
plot(1:max_iter+1, real(c_mean(index, :)));
hold on;
plot([1, max_iter+1], -real(a(index))*[1 1]);
title(['Real part of c_{mean}' int2str(index) ' and c_{opt}' int2str(index)]);
legend(['c' int2str(index)], ['a' int2str(index)]);
xlim([0 800])
xlabel('Number of iterations');

subplot(2, 1, 2);
plot(1:max_iter+1, imag(c_mean(index, :)));
hold on;
plot([1,max_iter+1], -imag(a(index))*[1 1]);
title(['Imaginary part of c_{mean}' int2str(index) ' and c_{opt}' int2str(index)]);
legend(['c' int2str(index)], ['a' int2str(index)]);
xlim([0 800])
xlabel('Number of iterations');
end

figure('Name','Mean squared error');
plot(1:max_iter,10*log10(mean_error), 1:max_iter, 10*log10(s_white)*ones(1, max_iter));
title('Mean squared error over iterations');
xlabel('Number of iterations'); ylabel('Mean Squared Error (dB)');


save('Jmin.mat', 'mean_error');
save('avg_coeff.mat', 'c_mean');
```

5 Various scripts

```
function [rx]=autocorrelation_Unb(x)
% Unbiased autocorrelation estimator
%INPUT: r.p. x, length of the autocorrelation Lcorr
%OUTPUT: autocorrelation estimate vector rx of length K=length(x)
%every index is augmented by 1 because matlab starts from 1 and not 0
K=length(x);

rx=zeros(K, 1);
for n=1:K
%first x that has k as argument
xnk=x(n:K);
%second x that has k-n as argument and the conjugate
xconj=conj(x(1:(K-n+1)));
rx(n)=(xnk.'*xconj)/(K-n+1);
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [corr] = correlogram( x, window, rx, L )
```

```
% Compute the PSD estimate using the correlogram method
%INPUT: r.p. x, window (hamming or rect, passed as a vector of samples
%starting from the origin), autocorrelation estimate rx, L number of
%samples used for the autocorrelation
%OUTPUT: corr, the PSD estimate using the correlogram method

K = length(x);
autoc_complete = full_autocorr(x, rx);
%computes the symmetric part of the window (negative samples)
full_win = zeros(K, 1);
%exploits periodicity instead of going to "negative indexes"
full_win(1 : L + 1) = window(L + 1 : 2*L + 1);
full_win(K - L + 1 : K) = window(1 : L);

windowed_autoc = autoc_complete .* full_win;
corr = fft(windowed_autoc);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ autoc_full ] = full_autocorr(x, rx)
% Finds the negative symmetrical part of the autocorrelation estimate
%INPUT: r.p. x, autocorrelation estimate rx
%OUTPUT: vector autoc_full with the autocorrelation (both negative and positive terms)

L=length(rx);
K = length(x);
% make autocorrelation simmetric
autoc_full = zeros(K, 1);
autoc_full(1:L) = rx;
temp = flipud(conj(rx));
% it's the same as puttig the conjugate, flipped, at the end of this
% vector, since fft is periodic
autoc_full((K-L+1):K) = temp(1:length(temp));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [a, sw, det_R]=findAR(N, rx)
%finds the parameters of an AR model of a r.p.
%INPUT: order of the filter N, autocorrelation of the r.p. rx
%OUTPUT: coefficients of the filter in the vector a, value of the variance
%and the determinant of R det_R to check for ill-conditioning
%of the white process

col=rx(1:N);
row=conj(col);
%construct the autocorrelation matrix NxN
R=toeplitz(col, row);
%compute the vector r
r=rx(2:N+1);
```

```
%compute the coefficients
det_R=det(R);
a=-inv(R)*r;
sw=abs(R(1,1)+r'*a);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [copt, Jmin]=predictor(rx, N)

[a, sw, det_R]=findAR(N, rx);
copt=-a;
Jmin=sw;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [welch_est, Ns] = welchPSD(inputsig, window, overlaps)
% Length of the window
D = length(window);
% Length of input signal
K = length(inputsig);
% Normalized energy of the window
Mw = sum(window .^ 2) * (1/D);
% Number of subsequences
N_s = floor((K-D)/(D-overlaps) + 1);
%Initialization of each periodogram
P_per = zeros(K, N_s);

for s = 0:(N_s-1)
% Windowed input
x_s = window .* inputsig(s*(D-overlaps)+1:s*(D-overlaps)+D);
% DFT on K samples of windowed input
X_s = fft(x_s, K);
% Periodogram for the window
P_per(:,s+1) = (abs(X_s)).^2 * (1/(D*Mw)); % Tc = 1;
end
% Sum of all periodograms
welch_est = sum(P_per, 2) * (1/N_s);
Ns = length(welch_est);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear global; clearvars; clc;

%% GENERATE THE PROCESS x(k), 300 REALIZATIONS

Nsamples=800;
% Frequencies of the exponentials
f1=0.17;
```

```
f2=0.78;
% Generate the white noise (2 components)
sigmaw=0.1;
x = zeros(800,300);

for i=1:300
% Real part
wi=sigmaw*randn(Nsamples,1);
% Imaginary part
wq=sigmaw*randn(Nsamples,1);
% Generate the initial phases
phi1=2*pi*rand(1);
phi2=2*pi*rand(1);

xi=zeros(Nsamples,1);
xq=zeros(Nsamples,1);
for k=1:Nsamples
xi(k)=cos(2*pi*f1*k+phi1)+0.8*cos(2*pi*f2*k+phi2)+wi(k);
xq(k)=sin(2*pi*f1*k+phi1)+0.8*sin(2*pi*f2*k+phi2)+wq(k);
end
x(:,i) = xi + 1i*xq;
end
save('realizations.mat', 'x');
```