Homework 8

Course Title: Digital Signal Processing I (Spring 2020)

Course Number: ECE53800

Instructor: Dr. Li Tan

Author: **Zhankun Luo**

Problems

9.1, 9.5, 9.9, 9.10, 9.16

Advanced Problems

9.30 (b),(d) (e), 9.31, 9.38

MATLAB

9.23, 9.24, 9.25, 9.28 [Bonus]

Problems

Problem 9.1

Given a quadratic MSE function for the Wiener filter:

$$J = 50 - 40w + 10w^2$$

find the optimal solution for w_* to achieve the minimum MSE $J_{\min}$ and determine $J_{\min}$.

solution

From the derivative of J(w)

$$\frac{\mathrm{d}J}{\mathrm{d}w}|_{w=w_*} = -40 + 20w|_{w=w_*} = -40 + 20w_* = 0$$

We conclude the extremum

$$w_* = 2, J(w)|_{w=w_*} = 10$$

Because for $\forall w \in R$, the second order derivative $\frac{\mathrm{d}^2 J}{\mathrm{d}w^2} = 20 > 0$

We conclude that $w_* = 2$ is the minima point, the min value $J_{\min} = J(w)|_{w=w_*} = 10$

Problem 9.5

Given a quadratic MSE function for the Wiener filter:

$$J = 50 - 40w + 10w^2$$

use the steepest decent method with an initial guess as w(0)=0 and the convergence factor μ=0.04 to find the optimal solution for w_* and determine $J_{\min}$ by iterating three times.

solution

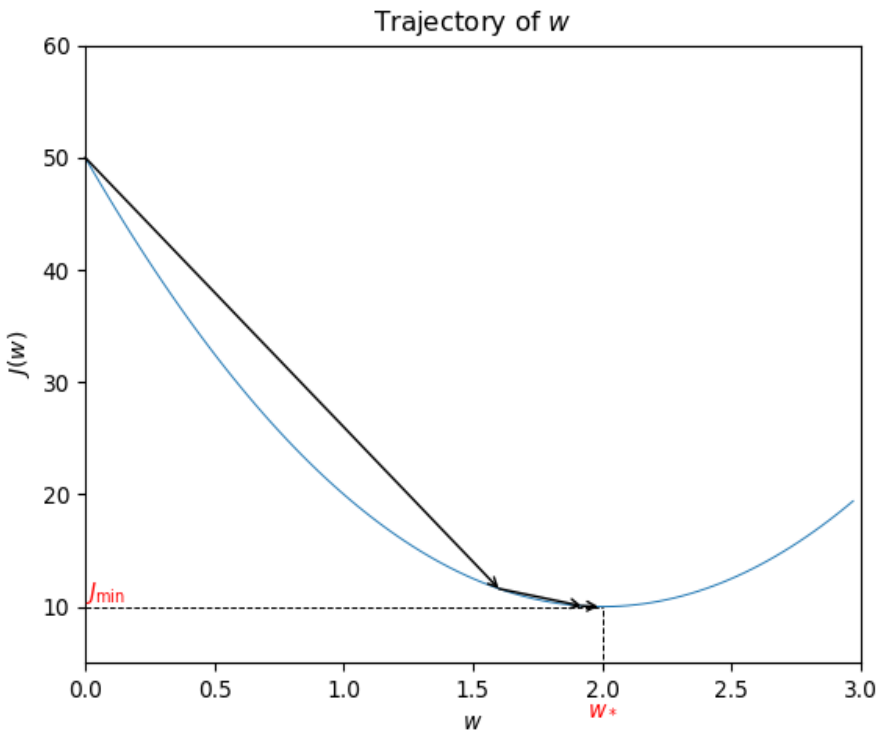
$$\frac{\mathrm{d}J}{\mathrm{d}w} = -40 + 20w$$

$$w \Leftarrow w - \mu\frac{\mathrm{d}J}{\mathrm{d}w}$$

After iterating three times, the $w, J(w)$ value are updated to be

$$w = [0, 1.6, 1.92, 1.984]$$
$$J(w) = [50, 11.6, 10.064, 10.00256]$$

Plot of the trajectory for w



Problem 9.9

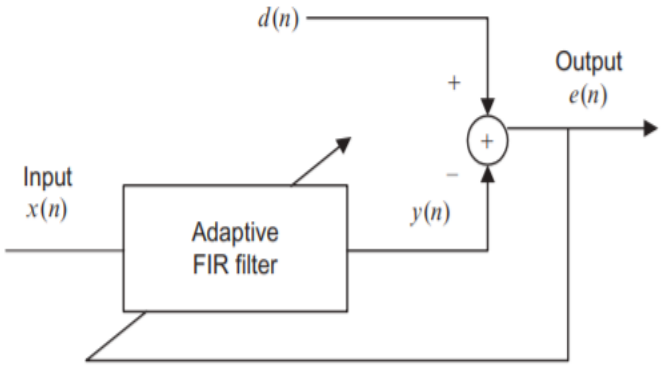


FIG. 9.22

Noise cancellation in Problem 9.9.

Given the following adaptive filter used for noise cancellation application (Fig. 9.22), in which d(0)=3, d(1)=2, d(2)=1, x(0)=3, x(1)=1, x(2)=2, and an adaptive filter with two taps: $y(n) = w_0 x(n) + w_1 x(n-1)$ with initial values $w_0 = 0, w_1 = 1$, and μ=0.1

(a) Determine the LMS algorithm equations

$y(n) =$
$e(n) =$
$w_0 =$
$w_1 =$

(b) Perform adaptive filtering for each n=0, 1, 2.

(c) Determine equations using the RLS algorithm.

(d) Repeat (b) using the RLS algorithm with δ=1/2 and λ=0.96.

solution

(a) Determine the LMS algorithm equations

$$y(n) = \sum_{k=0}^{1} w_k x(n-k) = w_0 x(n) + w_1 x(n-1)$$
$$e(n) = d(n) - y(n)$$
$$w_0 = w_0 + 2\mu e(n) x(n)$$
$$w_1 = w_1 + 2\mu e(n) x(n-1)$$

(b) Perform adaptive filtering for each n=0, 1, 2.

$$w = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1.8 \\ 1 \end{bmatrix}, \begin{bmatrix} 1.24 \\ -0.68 \end{bmatrix}, \begin{bmatrix} 0.92 \\ -0.84 \end{bmatrix} \quad [1\text{st: } n = -1]$$
$$y(n) = 0, 4.8, 1.8$$
$$e(n) = 3, -2.8, -0.8$$

Python script is below:

```
from rls.lms import Lms
from rls.rls import Rls
list_x = [3, 1, 2]
list_d = [3, 2, 1]
######################
# LMS adaptive filter
######################
N, mu, w_0 = 2, 0.1, [0, 1]
lms = Lms(mu, N, w_0)
list_y, list_e, list_w = [], [], [w_0]
for x, d in list(zip(list_x, list_d)):
    lms.train(x, d)
    list_y.append(lms.y)
    list_e.append(lms.e)
    list_w.append(lms.w)
print(list_y)
print(list_e)
print(list_w)
print('\n')
```

(c) Determine equations using the RLS algorithm. [initialization for $w_k = 0, Q(-1) = \delta \times I$]

$$X(n) \Leftarrow \begin{bmatrix} x(n) \\ x(n-1) \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\alpha(n) = d(n) - w^T X(n) = d(n) - \sum_{k=0}^{1} w_k x(n-k)$$

$$\vec{k}(n) = [\frac{1}{\lambda + X^T(n) Q(n-1) X(n)}] Q(n-1) X(n)$$

$$Q(n) \Leftarrow \frac{1}{\lambda}[Q(n-1) - \vec{k}(n) X^T(n) Q(n-1)]$$

$$w \Leftarrow w + \alpha(n) \vec{k}(n)$$

$$y(n) = w^T X(n) = \sum_{k=0}^{4} w_k x(n-k)$$

$$e(n) = d(n) - y(n)$$

(d) Repeat (b) using the RLS algorithm with δ=1/2 and λ=0.96.

$$Q(n) = \begin{bmatrix} 0.5 & 0.0 \\ 0.0 & 0.5 \end{bmatrix}, \begin{bmatrix} 0.091575 & 0.0 \\ 0.0 & 0.520833 \end{bmatrix}, \begin{bmatrix} 0.093869 & -0.025971 \\ -0.025971 & 0.099409 \end{bmatrix}, \begin{bmatrix} 0.0773 & -0.033062 \\ -0.033062 & 0.101788 \end{bmatrix}$$

$[1\text{st } \mathrm{Q(n)}: n = -1]$

$$k(n) = \begin{bmatrix} 0.274725 \\ 0.0 \end{bmatrix}, \begin{bmatrix} 0.015956 \\ 0.272256 \end{bmatrix}, \begin{bmatrix} 0.121538 \\ 0.035663 \end{bmatrix}$$

$$\alpha(n) = 3, 1.175824, -1.006001$$

$$w = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.824176 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.842938 \\ 0.320126 \end{bmatrix}, \begin{bmatrix} 0.720671 \\ 0.284249 \end{bmatrix} \quad [1\text{st: } n = -1]$$

$$y(n) = 2.472527, 1.803315, 1.725590$$

$$e(n) = 0.527473, 0.196685, -0.725590$$

Python script is below:

```
#####################
# RLS adaptive filter
#####################
delta, lambda_, N = 1/2, 0.96, 2
rls = Rls(delta, lambda_, N) # initial w = [0, ..., 0]
list_y, list_e, list_alpha, list_w,  = [], [], [], [[0] * N]
for x, d in list(zip(list_x, list_d)):
    rls.train(x, d)
    list_y.append(rls.y)
    list_e.append(rls.e)
    list_alpha.append(rls.alpha)
    list_w.append(rls.w)
    print(rls.Q)
    print(rls.k)
print('\n')
print(list_y)
print(list_e)
print(list_alpha)
print(list_w)
print('\n')
```

Problem 9.10

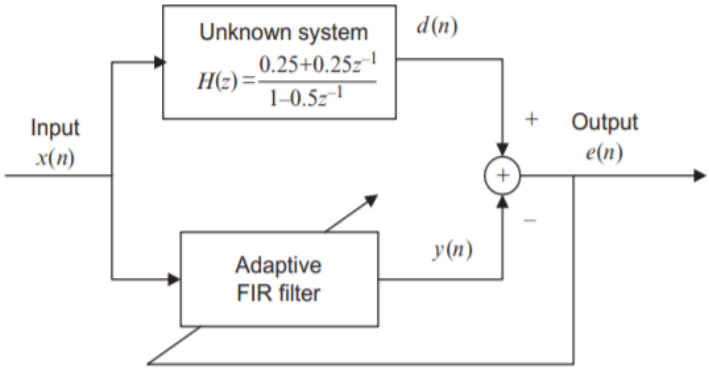


FIG. 9.23

System modeling in Problem 9.9.

Given a DSP system with a sampling rate set up to 8000 samples per second, implement adaptive filter with five taps for system modeling. As shown in Fig. 9.23, assume that the unknown system transfer function is

$$H(z) = \frac{0.25 + 0.25z^{-1}}{1 - 0.5z^{-1}}$$

(a) Determine the DSP equations using the LMS algorithm

$y(n) =$
$e(n) =$
$w_k =$

for i = 0, 1, 2, 3, 4; that is, write the equations for all adaptive coefficients:

$w_0 =$
$w_1 =$
$w_2 =$
$w_3 =$
$w_4 =$

(b) Determine equations using the RLS algorithm.

solution

(a) Determine the DSP equations using the LMS algorithm

$$y(n) = \sum_{k=0}^{4} w_k x(n-k)$$
$$e(n) = d(n) - y(n)$$
$$w_k \Leftarrow w_k + 2\mu e(n) x(n-k)$$

for i = 0, 1, 2, 3, 4; that is, write the equations for all adaptive coefficients:

$$w_0 = w_0 + 2\mu e(n) x(n)$$
$$w_1 = w_1 + 2\mu e(n) x(n-1)$$
$$w_2 = w_2 + 2\mu e(n) x(n-2)$$
$$w_3 = w_3 + 2\mu e(n) x(n-3)$$
$$w_4 = w_4 + 2\mu e(n) x(n-4)$$

(b) Determine equations using the RLS algorithm.

After the initialization for $w_k = 0, Q(-1) = \delta \times I$,

$$X(n) \Leftarrow \begin{bmatrix} x(n) \\ x(n-1) \\ x(n-2) \\ x(n-3) \\ x(n-4) \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

$$\alpha(n) = d(n) - w^T X(n) = d(n) - \sum_{k=0}^{4} w_k x(n-k)$$

$$\vec{k}(n) = [\frac{1}{\lambda + X^T(n) Q(n-1) X(n)}] Q(n-1) X(n)$$

$$Q(n) \Leftarrow \frac{1}{\lambda}[Q(n-1) - \vec{k}(n) X^T(n) Q(n-1)]$$

$$w \Leftarrow w + \alpha(n) \vec{k}(n)$$

$$y(n) = w^T X(n) = \sum_{k=0}^{4} w_k x(n-k)$$

$$e(n) = d(n) - y(n)$$

Problem 9.16

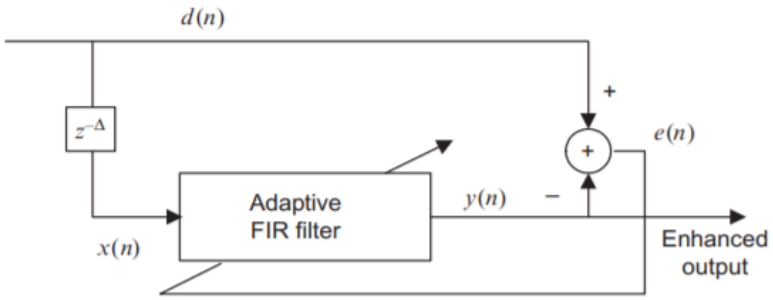


FIG. 9.26

Line enhancement in Problem 9.16.

For a line enhancement application using the FIR adaptive filter depicted in Fig. 9.26,

(a) Set up the LMS algorithm for the adaptive filter using two filter coefficients and delay Δ=2.

(b) Given the following inputs and outputs: d(0)=-1, d(1)=1, d(2)=-1, d(3)=1, d(4)=-1, d(5)=1 and d(6)=-1 and initial weights: $w_0 = w_1 = 0$, convergence factor is set to be μ=0.1, perform adaptive filtering to obtain outputs y(n) for n = 0, 1, 2, 3, 4.

(c) Determine equations using the RLS algorithm.

(d) Repeat (b) using the RLS algorithm with δ=1 and λ=0.96.

solution

This description is pretty ambiguous. There are **2 ways** of understanding:

1. use all the information of d(n)

$$d(-2), d(-1), d(0), d(1), d(2), d(3), d(4) = -1, 1, -1, 1, -1, 1, -1$$
$$x(0), x(1), x(2), x(3), x(4) = d(-2), d(-1), d(0), d(1), d(2) = -1, 1, -1, 1, -1$$

2. only use part information of d(n)

$$d(0), d(1), d(2), d(3), d(4), d(5), d(6) = -1, 1, -1, 1, -1, 1, -1$$
$$x(0), x(1), x(2), x(3), x(4) = d(-2), d(-1), d(0), d(1), d(2) = 0, 0, -1, 1, -1$$

(a) Set up the LMS algorithm for the adaptive filter using two filter coefficients and delay Δ=2.

$$y(n) = \sum_{k=0}^{1} w_k x(n-k) = w_0 x(n) + w_1 x(n-1)$$
$$e(n) = d(n) - y(n)$$
$$w_0 = w_0 + 2\mu e(n) x(n)$$
$$w_1 = w_1 + 2\mu e(n) x(n-1)$$

(b) Given the following inputs and outputs: d(0)=1, d(1)=1, d(2)=1, d(3)=1, d(4)=1, d(5)=1 and d(6)=1 and initial weights: $w_0 = w_1 = 0$, convergence factor is set to be μ=0.1, perform adaptive filtering to obtain outputs y(n) for n = 0, 1, 2, 3, 4.

1. use all the information of d(n)

$$d(-2), d(-1), d(0), d(1), d(2), d(3), d(4) = -1, 1, -1, 1, -1, 1, -1$$
$$x(0), x(1), x(2), x(3), x(4) = d(-2), d(-1), d(0), d(1), d(2) = -1, 1, -1, 1, -1$$

Then the results are

$$w = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.36 \\ -0.16 \end{bmatrix}, \begin{bmatrix} 0.456 \\ -0.256 \end{bmatrix}, \begin{bmatrix} 0.5136 \\ -0.3136 \end{bmatrix}, \begin{bmatrix} 0.54816 \\ -0.34816 \end{bmatrix} \quad [1\text{st: } n = -1]$$
$$y(n) = 0, 0.2, -0.52, 0.712, -0.8272$$
$$e(n) = -1, 0.8, -0.48, 0.288, -0.1728$$

2. only use part information of d(n)

$$d(0), d(1), d(2), d(3), d(4), d(5), d(6) = -1, 1, -1, 1, -1, 1, -1$$
$$x(0), x(1), x(2), x(3), x(4) = d(-2), d(-1), d(0), d(1), d(2) = 0, 0, -1, 1, -1$$

The results are

$$w = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.36 \\ -0.16 \end{bmatrix}, \begin{bmatrix} 0.456 \\ -0.256 \end{bmatrix} \quad [1\text{st: } n = -1]$$
$$y(n) = 0, 0, 0, 0.2, -0.52$$
$$e(n) = -1, 1, -1, 0.8, -0.48$$

(c) Determine equations using the RLS algorithm.

After the initialization for $w_k = 0, Q(-1) = \delta \times I$,

$$X(n) \Leftarrow \begin{bmatrix} x(n) \\ x(n-1) \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$
$$\alpha(n) = d(n) - w^T X(n) = d(n) - \sum_{k=0}^{1} w_k x(n-k)$$
$$\vec{k}(n) = [\frac{1}{\lambda + X^T(n) Q(n-1) X(n)}] Q(n-1) X(n)$$
$$Q(n) \Leftarrow \frac{1}{\lambda}[Q(n-1) - \vec{k}(n) X^T(n) Q(n-1)]$$
$$w \Leftarrow w + \alpha(n) \vec{k}(n)$$
$$y(n) = w^T X(n) = \sum_{k=0}^{4} w_k x(n-k)$$
$$e(n) = d(n) - y(n)$$

(d) Repeat (b) using the RLS algorithm with δ=1 and λ=0.96.

1. use all the information of d(n)

$$d(-2), d(-1), d(0), d(1), d(2), d(3), d(4) = -1, 1, -1, 1, -1, 1, -1$$
$$x(0), x(1), x(2), x(3), x(4) = d(-2), d(-1), d(0), d(1), d(2) = -1, 1, -1, 1, -1$$

The results are

$$Q(n) = \begin{bmatrix} 1 & 0.0 \\ 0.0 & 1 \end{bmatrix}, \begin{bmatrix} 0.510204 & 0.0 \\ 0.0 & 1.041667 \end{bmatrix}, \begin{bmatrix} 0.423513 & 0.220396 \\ 0.220396 & 0.635094 \end{bmatrix},$$
$$\begin{bmatrix} 0.413922 & 0.285189 \\ 0.285189 & 0.548019 \end{bmatrix}, \begin{bmatrix} 0.418397 & 0.323149 \\ 0.323149 & 0.517613 \end{bmatrix}, \begin{bmatrix} 0.428268 & 0.352052 \\ 0.352052 & 0.507659 \end{bmatrix}$$
$$[\text{1st Q(n): } n = -1]$$
$$k(n) = \begin{bmatrix} -0.510204 \\ 0.0 \end{bmatrix}, \begin{bmatrix} 0.203117 \\ -0.414698 \end{bmatrix}, \begin{bmatrix} -0.128733 \\ 0.262830 \end{bmatrix}, \begin{bmatrix} 0.095248 \\ -0.194464 \end{bmatrix}, \begin{bmatrix} -0.076216 \\ 0.155607 \end{bmatrix}$$
$$\alpha(n) = -1, 0.489796, -0.187193, 0.113895, -0.080898$$
$$w = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.510204 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.609690 \\ -0.203117 \end{bmatrix}, \begin{bmatrix} 0.633788 \\ -0.252317 \end{bmatrix}, \begin{bmatrix} 0.644636 \\ -0.274466 \end{bmatrix}, \begin{bmatrix} 0.650802 \\ -0.287054 \end{bmatrix} \quad [\text{1st: } n = -1]$$
$$y(n) = -0.510204, 0.812807, -0.886105, 0.919102, -0.937856$$
$$e(n) = -0.489796, 0.187193, -0.113895, 0.080898, -0.062144$$

2. only use part information of d(n)

$$d(0), d(1), d(2), d(3), d(4), d(5), d(6) = -1, 1, -1, 1, -1, 1, -1$$
$$x(0), x(1), x(2), x(3), x(4) = d(-2), d(-1), d(0), d(1), d(2) = 0, 0, -1, 1, -1$$

The results are

$$Q(n) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1.041667 & 0 \\ 0 & 1.041667 \end{bmatrix}, \begin{bmatrix} 1.085069 & 0 \\ 0 & 1.085069 \end{bmatrix},$$
$$\begin{bmatrix} 0.530578 & 0 \\ 0 & 1.130281 \end{bmatrix}, \begin{bmatrix} 0.440798 & 0.238353 \\ 0.238353 & 0.669617 \end{bmatrix}, \begin{bmatrix} 0.432377 & 0.305349 \\ 0.305349 & 0.575953 \end{bmatrix}$$
$$[\text{1st Q(n): } n = -1]$$
$$k(n) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -0.530578 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.202444 \\ -0.431263 \end{bmatrix}, \begin{bmatrix} -0.127027 \\ 0.270604 \end{bmatrix}$$
$$\alpha(n) = -1, 1, -1, 0.469422, -0.171945$$
$$w = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.530578 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.625610 \\ -0.202444 \end{bmatrix}, \begin{bmatrix} 0.647452 \\ -0.248974 \end{bmatrix} \quad [\text{1st: } n = -1]$$
$$y(n) = 0, 0, -0.530578, 0.828055, -0.896425$$
$$e(n) = -1, 1, -0.469422, 0.171945, -0.103575$$

Advanced Problems

Problem 9.30

(b) $E\left\{e^{jn\Omega}\right\}$

(d) $E\left\{(A\cos n\Omega)^2\right\}$

(e) $E\{(A\cos n\Omega)(A\sin n\Omega)\}$

where n is the time index, and $\Omega \neq 0$.

solution

(b) $E\left\{e^{jn\Omega}\right\}$

$$\begin{aligned} E\left\{e^{\mathrm{j}n\Omega}\right\} &= E\left\{\cos(n\Omega) + \mathrm{j}\sin(n\Omega)\right\} \\ &= E\left\{\cos(n\Omega)\right\} + \mathrm{j}E\left\{\sin(n\Omega)\right\} \\ &= 0 + \mathrm{j}0 = 0 \end{aligned}$$

(d) $E\left\{(A\cos n\Omega)^2\right\}$

$$\begin{aligned} E\left\{(A\cos n\Omega)^2\right\} &= E\left\{\frac{A^2}{2}[1+\cos(2n\Omega)]\right\} \\ &= \frac{A^2}{2}E\left\{1\right\} + \frac{A^2}{2}E\left\{\cos(2n\Omega)\right\} \\ &= \frac{A^2}{2} + 0 = \frac{A^2}{2} \end{aligned}$$

(e) $E\{(A\cos n\Omega)(A\sin n\Omega)\}$

$$\begin{aligned} E\{(A\cos n\Omega)(A\sin n\Omega)\} &= E\left\{\frac{A^2}{2}\sin(2n\Omega)\right\} \\ &= \frac{A^2}{2}E\left\{\sin(2n\Omega)\right\} \\ &= \frac{A^2}{2} \times 0 = 0 \end{aligned}$$

Problem 9.31

The following Wiener filter is used to predict the sinusoid:

$$d(n) = A\sin(n\Omega)$$

where the Wiener filter predictor with delays of n_1, n_2, and n_3 is given as

$$y(n) = w_1 d\left(n - n_1\right) + w_2 d\left(n - n_2\right) + w_3 d\left(n - n_3\right)$$

Find the Wiener filter coefficients: w_1, w_2, and w_3 .

solution

$$\begin{aligned} J &= E\left\{e^2(n)\right\} = E\left\{\left(d(n) - w^T X(n)\right)^2\right\} \\ &= w^T E\left\{X^T(n)X(n)\right\} w - 2E\{d(n)X(n)\}^T w + E\left\{d^2(n)\right\} \\ &= w^T R w - 2P^T w + \sigma^2 \end{aligned}$$

Here we define

$$\begin{aligned} w &\equiv \left[w_1 \ w_2 \ w_3\right]^T \\ X(n) &\equiv \begin{bmatrix} d(n-n_1) \\ d(n-n_2) \\ d(n-n_3) \end{bmatrix} \end{aligned}$$

Moreover, forR, P, σ^2, we conclude that (n_1, n_2, n_3 are not equal each other)

$$\begin{aligned} R &\equiv E\left\{X^T(n)X(n)\right\} \\ &= \begin{bmatrix} E\left\{d(n-n_1)d(n-n_1)\right\} & E\left\{d(n-n_1)d(n-n_2)\right\} & E\left\{d(n-n_1)d(n-n_3)\right\} \\ E\left\{d(n-n_2)d(n-n_1)\right\} & E\left\{d(n-n_2)d(n-n_2)\right\} & E\left\{d(n-n_2)d(n-n_3)\right\} \\ E\left\{d(n-n_3)d(n-n_1)\right\} & E\left\{d(n-n_3)d(n-n_2)\right\} & E\left\{d(n-n_3)d(n-n_3)\right\} \end{bmatrix} \\ P &\equiv E\{d(n)X(n)\} \\ &= \begin{bmatrix} E\left\{d(n)d(n-n_1)\right\} \\ E\left\{d(n)d(n-n_2)\right\} \\ E\left\{d(n)d(n-n_3)\right\} \end{bmatrix} \\ \sigma^2 &\equiv E\left\{d^2(n)\right\} \end{aligned}$$

Because we know that (for $i, j \in \{1, 2, 3\}$)

$$
\begin{aligned}
E\left\{d(n-n_i)d(n-n_j)\right\} &= E\left\{A^2\sin((n-n_i)\Omega)\sin((n-n_j)\Omega)\right\} \\
&= E\left\{\frac{A^2}{2}[\cos((n_j-n_i)\Omega)-\cos((2n-n_i-n_j)\Omega)]\right\} \\
&= E\left\{\frac{A^2}{2}\cos(|n_j-n_i|\Omega)\right\}-0 \\
&= \frac{A^2}{2}\cos(|n_j-n_i|\Omega) \\
&= \begin{cases} \frac{A^2}{2} & i=j \\ \frac{A^2}{2}\cos(|n_j-n_i|\Omega) & i\neq j \end{cases} \\
E\left\{d(n)d(n-n_i)\right\} &= E\left\{A^2\sin(n\Omega)\sin((n-n_i)\Omega)\right\} \\
&= E\left\{\frac{A^2}{2}[\cos(n_i\Omega)-\cos((2n-n_i)\Omega)]\right\} \\
&= E\left\{\frac{A^2}{2}\cos(n_i\Omega)\right\}-0 \\
&= \frac{A^2}{2}\cos(n_i\Omega) \\
E\left\{d^2(n)\right\} &\equiv E\left\{[A\sin(n\Omega)]^2\right\} \\
&= E\left\{\frac{A^2}{2}[1-\cos(2n\Omega)]\right\} \\
&= \frac{A^2}{2}-0=\frac{A^2}{2}
\end{aligned}
$$

Thus for R, P, σ^2, we conclude

$$
\begin{aligned}
R &= \begin{bmatrix} E\left\{d(n-n_1)d(n-n_1)\right\} & E\left\{d(n-n_1)d(n-n_2)\right\} & E\left\{d(n-n_1)d(n-n_3)\right\} \\ E\left\{d(n-n_2)d(n-n_1)\right\} & E\left\{d(n-n_2)d(n-n_2)\right\} & E\left\{d(n-n_2)d(n-n_3)\right\} \\ E\left\{d(n-n_3)d(n-n_1)\right\} & E\left\{d(n-n_3)d(n-n_2)\right\} & E\left\{d(n-n_3)d(n-n_3)\right\} \end{bmatrix} \\
&= \begin{bmatrix} \frac{A^2}{2}\cos(|n_1-n_1|\Omega) & \frac{A^2}{2}\cos(|n_1-n_2|\Omega) & \frac{A^2}{2}\cos(|n_1-n_3|\Omega) \\ \frac{A^2}{2}\cos(|n_2-n_1|\Omega) & \frac{A^2}{2}\cos(|n_2-n_2|\Omega) & \frac{A^2}{2}\cos(|n_2-n_3|\Omega) \\ \frac{A^2}{2}\cos(|n_3-n_1|\Omega) & \frac{A^2}{2}\cos(|n_3-n_2|\Omega) & \frac{A^2}{2}\cos(|n_3-n_3|\Omega) \end{bmatrix} \\
&= \frac{A^2}{2}\begin{bmatrix} 1 & \cos(|n_1-n_2|\Omega) & \cos(|n_1-n_3|\Omega) \\ \cos(|n_1-n_2|\Omega) & 1 & \cos(|n_2-n_3|\Omega) \\ \cos(|n_1-n_3|\Omega) & \cos(|n_2-n_3|\Omega) & 1 \end{bmatrix}
\end{aligned}
$$

$$
P = \begin{bmatrix} E\left\{d(n)d(n-n_1)\right\} \\ E\left\{d(n)d(n-n_2)\right\} \\ E\left\{d(n)d(n-n_3)\right\} \end{bmatrix} = \begin{bmatrix} \frac{A^2}{2}\cos(n_1\Omega) \\ \frac{A^2}{2}\cos(n_2\Omega) \\ \frac{A^2}{2}\cos(n_3\Omega) \end{bmatrix} = \frac{A^2}{2}\begin{bmatrix} \cos(n_1\Omega) \\ \cos(n_2\Omega) \\ \cos(n_3\Omega) \end{bmatrix}
$$

$$
\sigma^2 = E\left\{d^2(n)\right\} = \frac{A^2}{2}
$$

Find the Wiener filter coefficients w_* , here **if R is invertible**

$$w_* = R^{-1}P = \begin{bmatrix} 1 & \cos(|n_1-n_2|\Omega) & \cos(|n_1-n_3|\Omega) \\ \cos(|n_1-n_2|\Omega) & 1 & \cos(|n_2-n_3|\Omega) \\ \cos(|n_1-n_3|\Omega) & \cos(|n_2-n_3|\Omega) & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \cos(n_1\Omega) \\ \cos(n_2\Omega) \\ \cos(n_3\Omega) \end{bmatrix}$$

But **Unfortunately**, $\det(R) \equiv 0$ always holds, R^{-1} **Not Exist** . we verify it with Python script

```
import sympy as sym
from sympy import cos, sin
n1, n2, n3, omega = sym.symbols('n_1, n_2, n_3, omega')
R = sym.Matrix([[1, cos((n1-n2)*omega), cos((n1-n3)*omega)],
                [cos((n1-n2)*omega), 1, cos((n2-n3)*omega)],
                [cos((n1-n3)*omega), cos((n2-n3)*omega), 1]])
P = sym.Matrix([[cos(n1*omega)],
                [cos(n2*omega)],
                [cos(n3*omega)]])
det = R.det()
sym.trigsimp(sym.cancel(det)) # symplify the form of det:= 0
```

Now we want to find all solutions $w_* = w_{\text{special}} + w_{\text{basic}}$ for equation $Rw_* = P$

Where w_{special} is special solution for $Rw_* = P$

Here w_{basic} is set of basic solution for $Rw_* = 0$

$[R \quad P]$ elementary row operations(初等行变换)

```
RP = R.col_insert(3, sym.Matrix([cos(n1*omega), cos(n2*omega), cos(n3*omega)]))
rref = sym.trigsimp(RP.rref()) # [R P] elementary row operations && symplify form
print(sym.latex(rref))
```

$$[R \quad P] = \begin{bmatrix} 1 & \cos(\Omega(n_1-n_2)) & \cos(\Omega(n_1-n_3)) & \cos(\Omega n_1) \\ \cos(\Omega(n_1-n_2)) & 1 & \cos(\Omega(n_2-n_3)) & \cos(\Omega n_2) \\ \cos(\Omega(n_1-n_3)) & \cos(\Omega(n_2-n_3)) & 1 & \cos(\Omega n_3) \end{bmatrix}$$

$$\xrightarrow{\text{elementary row operations}} \begin{bmatrix} 1 & 0 & \frac{\cos(\Omega(n_1-n_3))-\cos(\Omega(n_1-2n_2+n_3))}{2\sin^2(\Omega(n_1-n_2))} & -\frac{\sin(\Omega n_2)}{\sin(\Omega(n_1-n_2))} \\ 0 & 1 & \frac{\cos(\Omega(n_2-n_3))-\cos(\Omega(-2n_1+n_2+n_3))}{2\sin^2(\Omega(n_1-n_2))} & \frac{\sin(\Omega n_1)}{\sin(\Omega(n_1-n_2))} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Typically, we can find w_{special} is special solution for $Rw_* = P$

$$w_{\text{special}} = \begin{bmatrix} -\frac{\sin(\Omega n_2)}{\sin(\Omega(n_1-n_2))} \\ \frac{\sin(\Omega n_1)}{\sin(\Omega(n_1-n_2))} \\ 0 \end{bmatrix}$$

The w_{basic} set of basic solution for $Rw_* = 0$ are $[\forall k \in R, k' = -2\sin(n_1-n_2)k]$

$$
\begin{aligned}
w_{\text{basic}} &= k \cdot \begin{bmatrix} \cos\left(\Omega\left(n_1 - n_3\right)\right) - \cos\left(\Omega\left(n_1 - 2n_2 + n_3\right)\right) \\ \cos\left(\Omega\left(n_2 - n_3\right)\right) - \cos\left(\Omega\left(-2n_1 + n_2 + n_3\right)\right) \\ -2\sin^2\left(\Omega\left(n_1 - n_2\right)\right) \end{bmatrix} \\
&= k' \cdot \begin{bmatrix} \sin\left(\Omega\left(n_2 - n_3\right)\right) \\ \sin\left(\Omega\left(n_3 - n_1\right)\right) \\ \sin\left(\Omega\left(n_1 - n_2\right)\right) \end{bmatrix}
\end{aligned}
$$

Finally, we find all solutions $w_* = w_{\text{special}} + w_{\text{basic}}$ for equation $Rw_* = P$

$$
w_* = w_{\text{special}} + w_{\text{basic}} = \begin{bmatrix} -\frac{\sin\left(\Omega n_2\right)}{\sin\left(\Omega(n_1 - n_2)\right)} \\ \frac{\sin\left(\Omega n_1\right)}{\sin\left(\Omega(n_1 - n_2)\right)} \\ 0 \end{bmatrix} + k' \cdot \begin{bmatrix} \sin\left(\Omega\left(n_2 - n_3\right)\right) \\ \sin\left(\Omega\left(n_3 - n_1\right)\right) \\ \sin\left(\Omega\left(n_1 - n_2\right)\right) \end{bmatrix}
$$

Problem 9.38

Given the matrix inversion lemma: $A^{-1} = B - BC\left(D + C^T BC\right)^{-1} C^T B$

Verify the matrix inversion lemma using the following:

$$A = \begin{bmatrix} 2 & -1/2 \\ 0 & 1/2 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad D = 1$$

solution

From Woodbury matrix identity

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1} VA^{-1}$$

short proof

> Consider equation, here X is inverse of (A+U C V)
>
> $$\begin{aligned} &(A + UCV)X = I \\ \Leftrightarrow &\begin{cases} AX + UY = I \\ VX - C^{-1}Y = 0 \end{cases} \end{aligned}$$
>
> For 1st equation, use Y to represent X
>
> $$\begin{aligned} &X = A^{-1}(I - UY) \\ \Rightarrow &VX = VA^{-1}(I - UY) \end{aligned}$$
>
> Substitute V X to 2nd equation
>
> $$\begin{aligned} &VA^{-1}(I - UY) - C^{-1}Y = 0 \\ \Rightarrow &VA^{-1} = [VA^{-1}U + C^{-1}]Y \\ \Rightarrow &[VA^{-1}U + C^{-1}]^{-1} VA^{-1} = Y \end{aligned}$$
>
> Replace Y in the 1st equation
>
> $$\begin{aligned} &AX + U[VA^{-1}U + C^{-1}]^{-1} VA^{-1} = I \\ \Rightarrow &AX = I - U[VA^{-1}U + C^{-1}]^{-1} VA^{-1} \end{aligned}$$
>
> Finally
>
> $$\begin{aligned} X &= A^{-1}\left[I - U[VA^{-1}U + C^{-1}]^{-1} VA^{-1}\right] \\ &= A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1} VA^{-1} \end{aligned}$$

Then replace symbols

$$A \Leftarrow B^{-1}$$
$$U \Leftarrow C$$
$$C \Leftarrow D$$
$$V \Leftarrow C^T$$

For inverse of A, we have

$$A = (B^{-1} + CDC^T)$$
$$\Rightarrow A^{-1} = B - BC\big(D + C^T BC\big)^{-1} C^T B$$

Example

$$A = \begin{bmatrix} 2 & -1/2 \\ 0 & 1/2 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad D = 1$$

For the left side

$$LS = A^{-1} = \begin{bmatrix} 2 & -1/2 \\ 0 & 1/2 \end{bmatrix}^{-1} = \frac{\begin{bmatrix} 1/2 & 1/2 \\ 0 & 2 \end{bmatrix}}{2 \times \frac{1}{2} - 0 \times (-\frac{1}{2})} = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 2 \end{bmatrix}$$

For the right side

$$RS = B - BC\big(D + C^T BC\big)^{-1} C^T B$$
$$= \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} (1 + \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix})^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} 2^{-1} [1 \quad 1]$$
$$= \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} - (1/2) \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 2 \end{bmatrix}$$

So, it is verified that $LS = RS$, thus yields

$$A^{-1} = B - BC\big(D + C^T BC\big)^{-1} C^T B$$

MATLAB Projects

Problem 9.23

Write a MATLAB program for minimizing the two-weight mean squared error (MSE) function

$$J = 100 + 100w_1^2 + 4w_2^2 - 100w_1 - 8w_2 + 10w_1w_2$$

by applying the steepest descent algorithm for 500 iterations. The derivatives are derived as

$$\frac{dJ}{dw_1} = 200w_1 - 100 + 10w_2, \quad \frac{dJ}{dw_2} = 8w_2 - 8 + 10w_1$$

and the initial weights are assumed as w_1(0)=0, w_2(0)=0, μ=0.001. Plot w_1(k), w_2(k), and J(k) vs. the number of iterations, respectively, and summarize your results.

solution

$$\begin{aligned} J &= w^T \begin{bmatrix} 100 & 5 \\ 5 & 4 \end{bmatrix} w - 2 \times \begin{bmatrix} 50 \\ 4 \end{bmatrix}^T w + 100 \\ &= w^T R w - 2P^T w + \sigma^2 \end{aligned}$$

Here, $w = [w_1, w_2]^T$, and $R = \begin{bmatrix} 100 & 5 \\ 5 & 4 \end{bmatrix}, P = \begin{bmatrix} 50 \\ 4 \end{bmatrix}, \sigma^2 = 100$, thus

$$\nabla J = 2(Rw - P) = \begin{bmatrix} 200w_1 - 100 + 10w_2 \\ 8w_2 - 8 + 10w_1 \end{bmatrix}$$

R is positive definite, when $\nabla J = 0, w = w_*$, J is the minima

$$w_* = R^{-1}P = \frac{\begin{bmatrix} 4 & -5 \\ -5 & 100 \end{bmatrix}}{100 \times 4 - 5^2} \cdot \begin{bmatrix} 50 \\ 4 \end{bmatrix} = \frac{\begin{bmatrix} 180 \\ 150 \end{bmatrix}}{375} = \begin{bmatrix} \frac{12}{25} \\ \frac{2}{5} \end{bmatrix} = \begin{bmatrix} 0.48 \\ 0.4 \end{bmatrix}$$

$$J(w_*) = -P^T w_* + \sigma^2 = \begin{bmatrix} 50 \\ 4 \end{bmatrix}^T \begin{bmatrix} 0.48 \\ 0.4 \end{bmatrix} + 100 = 74.4$$

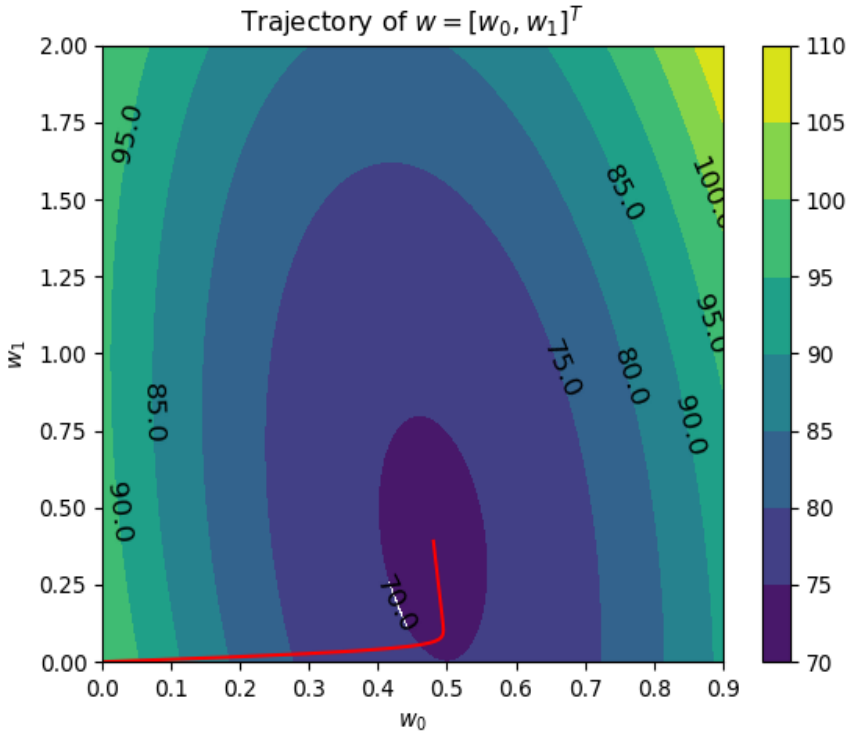
The steepest descent algorithm

$$w \Leftarrow w - \mu \nabla J$$

Finally, after 500 iterations:

$$\begin{aligned} w_* &= [0.480455, 0.391241] \\ J(w_*) &= 74.4003 \end{aligned}$$

Plot Trajectory of w



Python code is below

```
try:
    from rls.matrix import Matrix
except ModuleNotFoundError:
    from matrix import Matrix

class Quad(object):
    def __init__(self, R, P, sq_sigma, mu, w_0=None):
        if isinstance(R, Matrix):
            self.R = R
        elif isinstance(R, list):
            row = len(R)
            col = len(R[0])
            if row != col:
                raise Exception("R must be n x n matrix")
            self.R = Matrix(R)
        if self.R.T() != self.R:
            raise Exception("R must by symmetric amtrix")
        if len(R) != len(P):
            raise Exception("R, P must have the same length")
        self.P = P
        self.sq_sigma = sq_sigma
        self.mu = mu
        if w_0 == None:
            w_0 = [0] * len(P)
        self.w = w_0
        self.__update_grad()
```

```
    def __update_grad(self):
        temp = (Matrix([self.w]) * self.R)[0]
        self.grad = [2*(t_ - p_) for t_, p_ in list(zip(temp, self.P))]

    def train(self):
        self.w = [w_ - self.mu * grad_ for w_, grad_ in list(zip(self.w, self.grad))]
        self.__update_grad()

    def eval(self, w_eval=None):
        if w_eval == None:
            w_eval = self.w # default func(self.w)
        if len(w_eval) != len(self.w):
            raise Exception("w_eval and self.w have the same length")
        temp = (Matrix([w_eval]) * self.R)[0]
        temp = [temp_ - 2 * p_ for temp_, p_ in list(zip(temp, self.P))]
        return sum([temp_ * w_ for temp_, w_ in list(zip(temp,  w_eval))]) \
            + self.sq_sigma

if __name__ == "__main__":
    R, P, sq_sigma = [[100, 5], [5, 4]], [50, 4], 100
    mu, w_0 = 0.001, [0, 0]
    quad = Quad(R, P, sq_sigma, mu, w_0)
    list_w = [w_0]
    for ind in range(500):
        quad.train()
        list_w.append( quad.w )
    list_w = list(map(list, zip(*list_w))) # transpose: (n, 2) => (2, n)
    print(quad.w)  # w*: minima point
    print(quad.eval()) # J(w*): min value
    ########################################
    # plot Trajectory of w
    ########################################
    import matplotlib.pyplot as plt
    import numpy as np
    xlist = np.linspace(0, 0.9, 100)
    ylist = np.linspace(0, 2, 100)
    X, Y = np.meshgrid(xlist, ylist)
    row, col = len(ylist), len(xlist)
    Z = [[ quad.eval([X[n_row][n_col], Y[n_row][n_col]]) for n_col in range(col)] \
        for n_row in range(row)]
    fig, ax = plt.subplots(1, 1)
    cp = ax.contourf(X, Y, Z)
    ax.clabel(cp, colors = 'k', fmt = '%2.1f', fontsize=12)
    fig.colorbar(cp) # Add a colorbar to a plot
    ax.set_title(r'Trajectory of $w=[w_0, w_1]^T$')
    ax.set_xlabel(r'$w_0$')
    ax.set_ylabel(r'$w_1$')
    ax.plot(list_w[0], list_w[1], 'r')
    fig.savefig("../p9_23.png")
    plt.show()
```

Problem 9.24

In Problem 9.10, the unknown system is assumed as a fourth-order Butterworth bandpass filter with a lower cutoff frequency of 700 Hz and an upper cutoff frequency of 900 Hz. Design a bandpass filter by the bilinear transformation method for simulating the unknown system with a sampling rate of 8000 Hz.

(a) Generate the input signal for 0.1 s using a sum of three sinusoids having 100, 800, and 1500 Hz with a sampling rate of 8000Hz.

(b) Use the generated input as the unknown system input to produce the system output. The adaptive FIR filter is then applied to model the designed bandpass filter. The following parameters are assumed:

- Adaptive FIR filter
- Number of taps: 15 coefficients
- Algorithm: LMS algorithm
- Convergence factor: 0.01

(c) Implement the adaptive FIR filter, plot the system input, system output, adaptive filter output, and the error signal, respectively.

(d) Plot the input spectrum, system output spectrum, and adaptive filter output spectrum, respectively.

(e) Repeat (a)–(d) using the RLS algorithm with δ=1 and λ=0.96.

solution

(a) Generate the input signal for 0.1 s using a sum of three sinusoids having 100, 800, and 1500 Hz with a sampling rate of 8000Hz.

```
f_sample, list_f = 8000, [100, 800, 1500]
length = ceil(0.1 * f_sample)  # duration 0.1 second
list_x = [sum([sin(2*pi * (f/f_sample) * ind) for f in list_f]) for ind in
range(length)]
```

(b) Use the generated input as the unknown system input to produce the system output. Unknown system: a **fourth-order Butterworth bandpass** filter

$\omega_{zl} = 2\pi \times 700, \omega_{zh} = 2\pi \times 900$ rad/s

$\omega_{sl} = 2f_s \tan(\frac{\omega_{zl}}{2f_s}), \omega_{sh} = 2f_s \tan(\frac{\omega_{zh}}{2f_s})$, so, ω_{sl}, ω_{sh} = [4512.4667, 5902.7116]

Then $w_0 = \sqrt{\omega_{sl} \times \omega_{sh}}, W = \omega_{sh} - \omega_{sl}$, then

The order of Butterworth prototype: 4 / 2 = 2 (band pass)

$$H(s') = \frac{1}{s'^2 + 1.4142s' + 1}$$

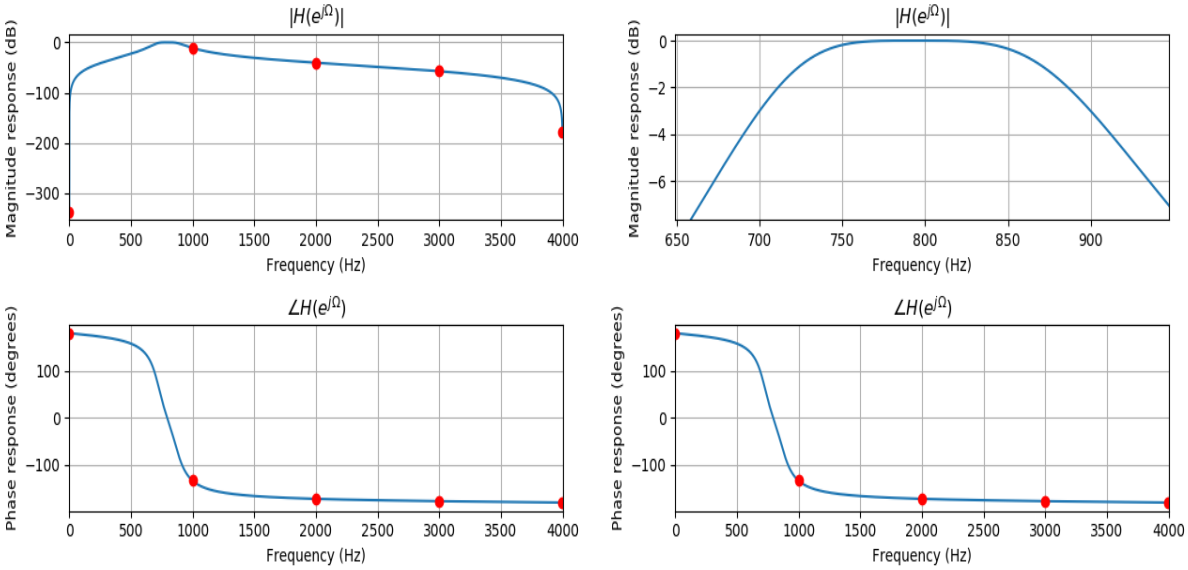
Substitute $s' = \frac{s^2+\omega_0^2}{sW}$ for band pass filter, thus

$$H(s) = H(s')|_{s'=\frac{s^2+\omega_0^2}{sW}} = \frac{1932780.9958s^2}{s^4 + 1966.1033s^3 + 55204360.2852s^2 + 52368712599.4459s + 709465289998621.9}$$

At last, BLT $s = 2f_s \frac{1-z^{-1}}{1+z^{-1}}$ to yield

$$H(z) = H(s)\big|_{s=2f_s \frac{1-z^{-1}}{1+z^{-1}}} = \frac{0.0055 - 0.0111z^{-2} + 0.0055z^{-4}}{1 - 3.0664z^{-1} + 4.1359z^{-2} - 2.7431z^{-3} + 0.8008z^{-4}}$$

The magnitude and phase plots for the unknown system

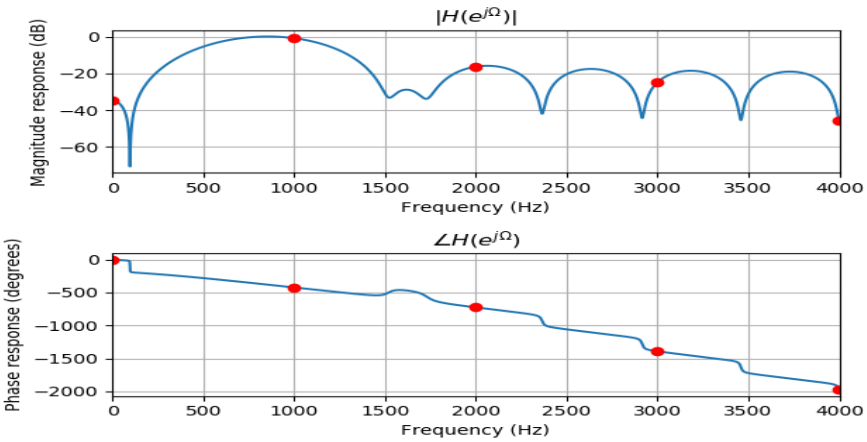


(c) Implement the adaptive FIR filter, plot the system input, system output, adaptive filter output, and the error signal, respectively.

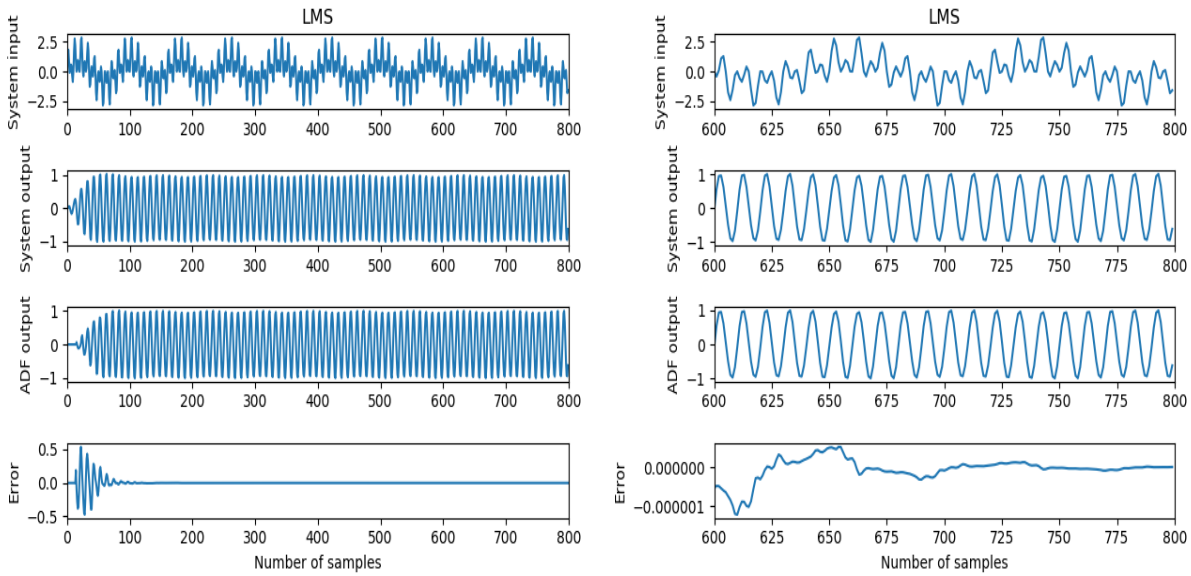
- Adaptive FIR filter
- Number of taps: 15 coefficients
- Algorithm: LMS algorithm
- Convergence factor: 0.01

The magnitude and phase plots for the **LMS** adaptive filter

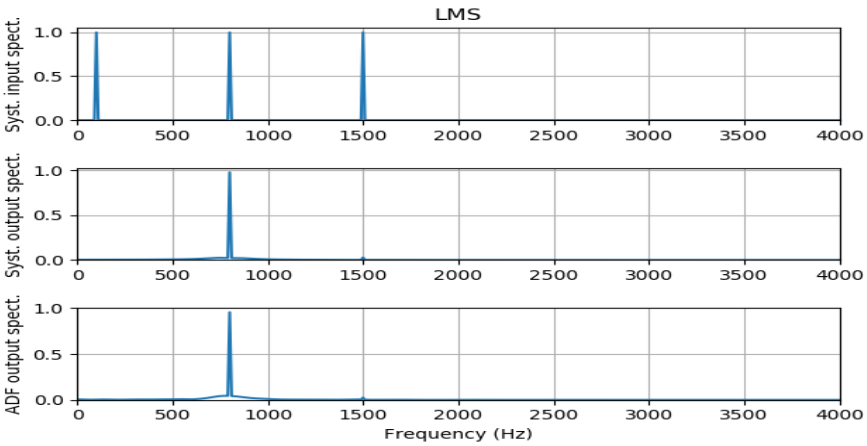
$$\begin{aligned}\left[w_0, \cdots, w_{N-1}\right] =& [0.098584, 0.081271, 0.052333, -0.010238, -0.104861, -0.179288, -0.170288, -0.070568, \\ & 0.055421, 0.128914, 0.125045, 0.078709, 0.030333, -0.017251, -0.080168]\end{aligned}$$



Plots of system input, system output, adaptive filter output, and the error signal.



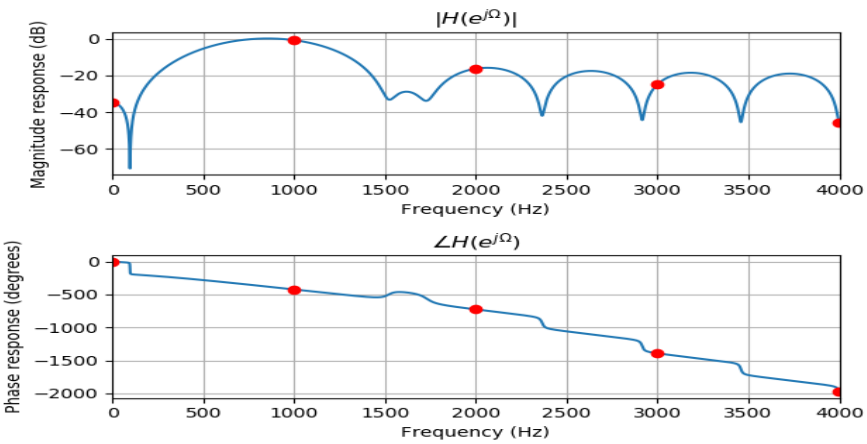
(d) Plot the input spectrum, system output spectrum, and adaptive filter output spectrum, respectively.



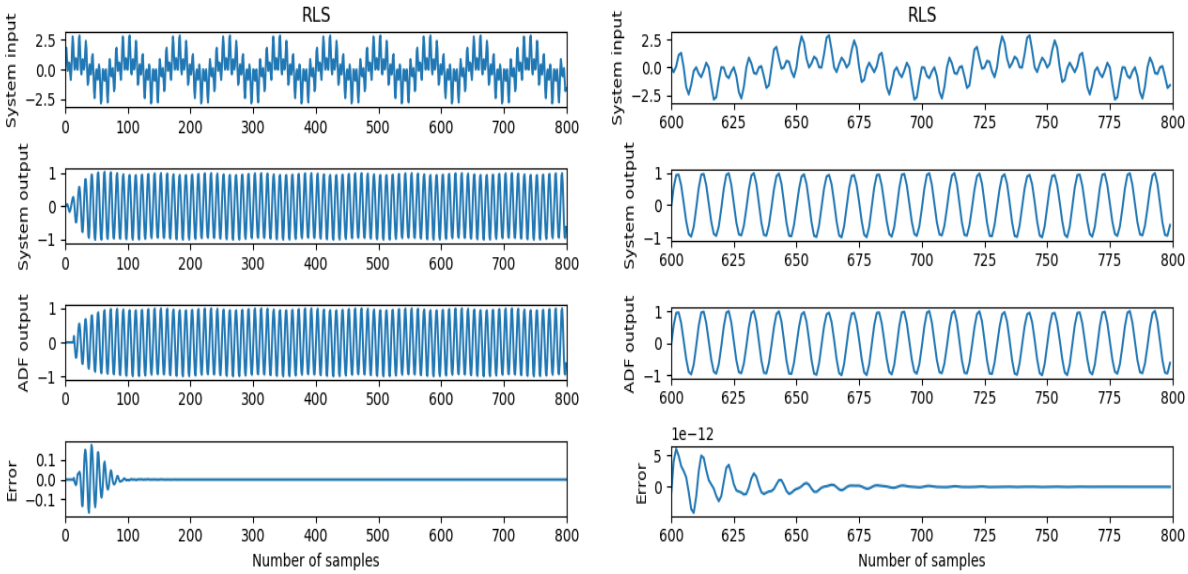
(e) Repeat (a)–(d) using the RLS algorithm with δ=1 and λ=0.96.

The magnitude and phase plots for the **RLS** adaptive filter

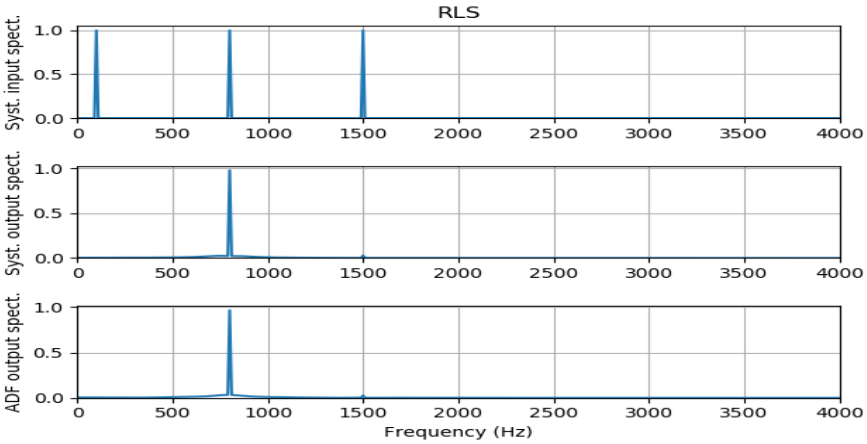
$$\left[w_0, \cdots, w_{N-1}\right] = [0.098584, 0.081271, 0.052333, -0.010238, -0.104861, -0.179288, -0.170288, -0.070568, 0.055421, 0.128914, 0.125045, 0.078709, 0.030333, -0.017251, -0.080168]$$



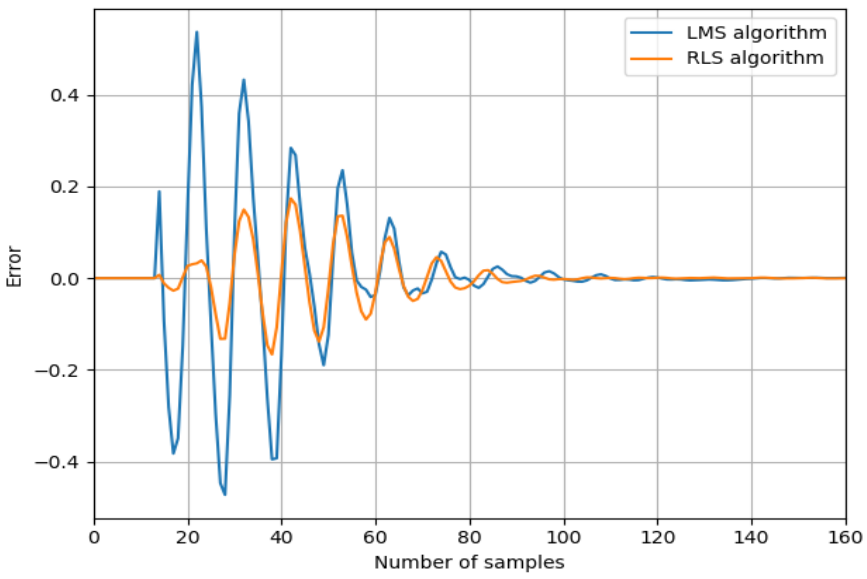
Plot of system input, system output, adaptive filter output, and the error signal.



Plots of the input spectrum, system output spectrum, and adaptive filter output spectrum.



Comparison for Errors of **LMS** and **RLS**



Problem 9.25

Use the following MATLAB code to generate the reference noise and the signal of 300 Hz corrupted by the noise with a sampling rate of 8000 Hz.

```
fs = 8000; T = 1/fs; % Sampling rate and sampling period
t = 0:T:1; % Create time instants
x = randn(1, length(t)); % Generate reference noise
n = filter([ 0 0 0 0 0 0 0 0 0 0.8 ], 1, x); % Generate the corruption noise
d = sin(2*pi * 300 * t) + n; % Generate the corrupted signal
```

(a) Implement an adaptive FIR filter to remove the noise. The adaptive filter specifications are as follows:

- Sample rate = 8000 Hz
- Signal corrupted by Gaussian noise delayed by nine samples from the reference noise
- Reference noise: Gaussian noise with a power of 1
- Number of FIR filter tap: 16
- Convergence factor for the LMS algorithm: 0.01

(b) Plot the corrupted signal, reference noise, and enhanced signal, respectively.

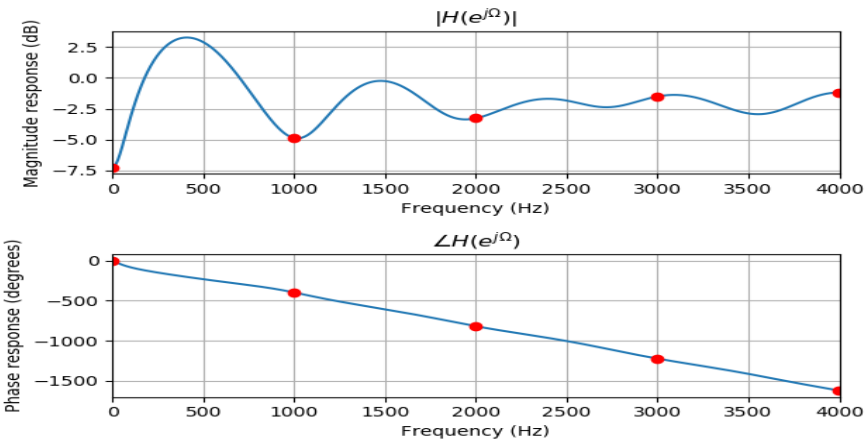
(c) Compare the spectral plots between the corrupted signal and the enhanced signal.

(d) Repeat (a)–(c) using the RLS algorithm with δ=1 and λ=0.96.

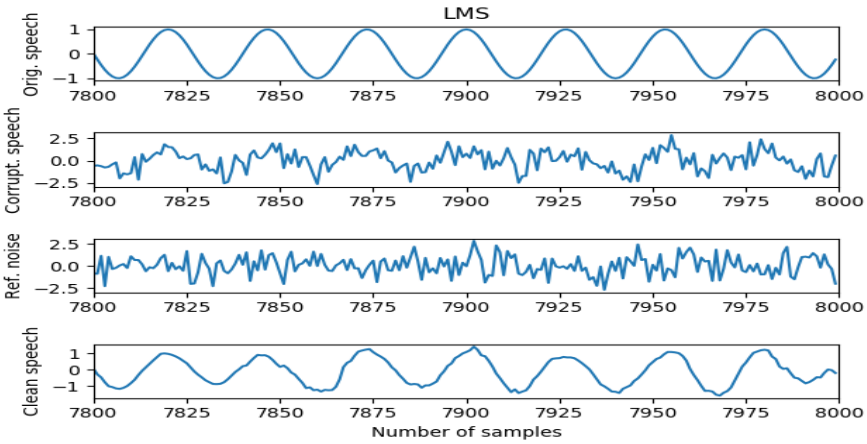
solution

1. Implement an adaptive FIR filter to remove the noise. The adaptive filter specifications are as follows:
    - Sample rate = 8000 Hz
    - Signal corrupted by Gaussian noise delayed by nine samples from the reference noise
    - Reference noise: Gaussian noise with a power of 1
    - Number of FIR filter tap: 16
    - Convergence factor for the LMS algorithm: 0.01

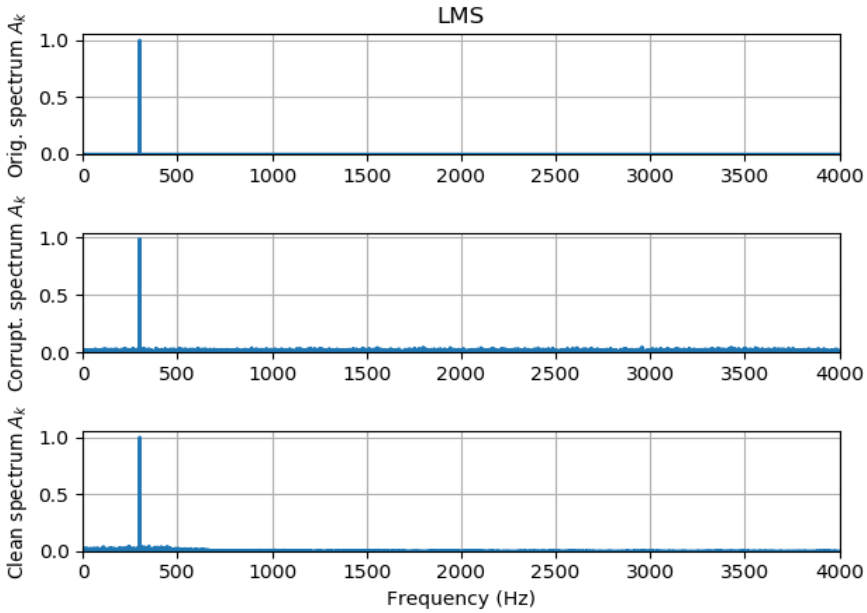
$$\left[w_0, \cdots, w_{N-1}\right] = [-0.117458, -0.120189, -0.116545, -0.118012, -0.09756, -0.087345, -0.067977, -0.046143, -0.012461, 0.803577, 0.041275, 0.067751, 0.068472, 0.070101, 0.082823, 0.083464]$$



2. Plot the corrupted signal, reference noise, and enhanced signal, respectively.



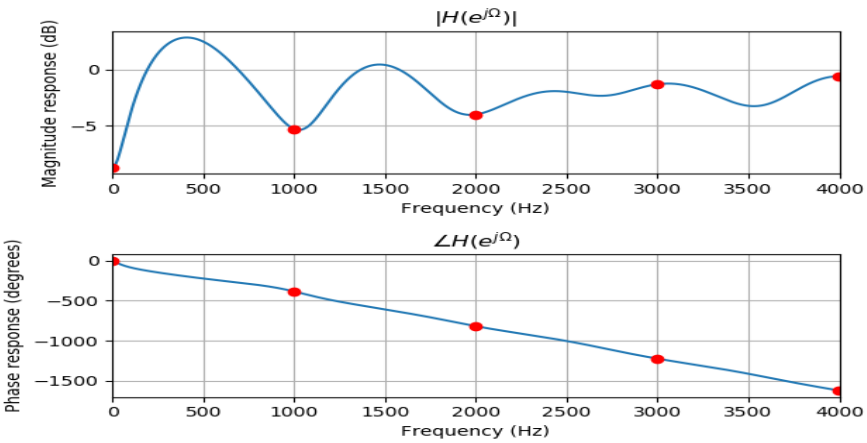
3. Compare the spectral plots between the corrupted signal and the enhanced signal.



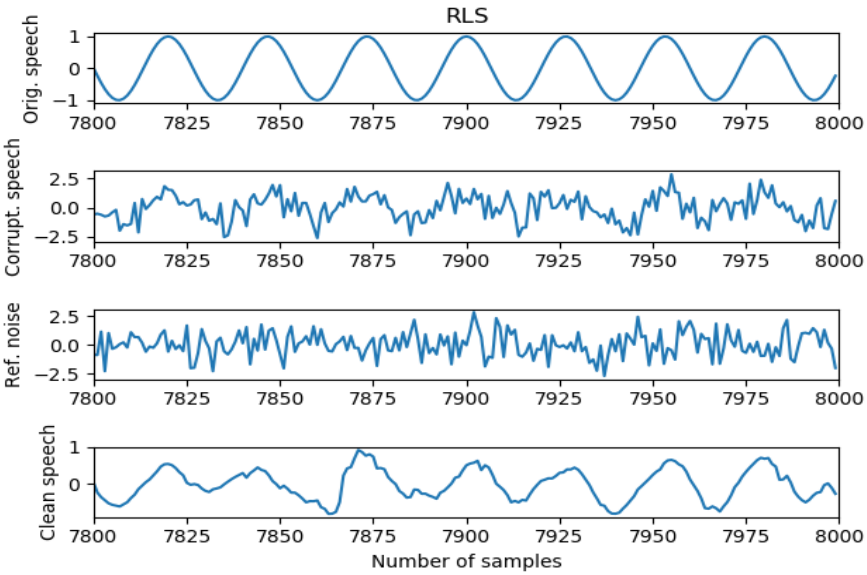
4. Repeat (a)–(c) using the RLS algorithm with δ=1 and λ=0.96.

The adaptive FIR filter

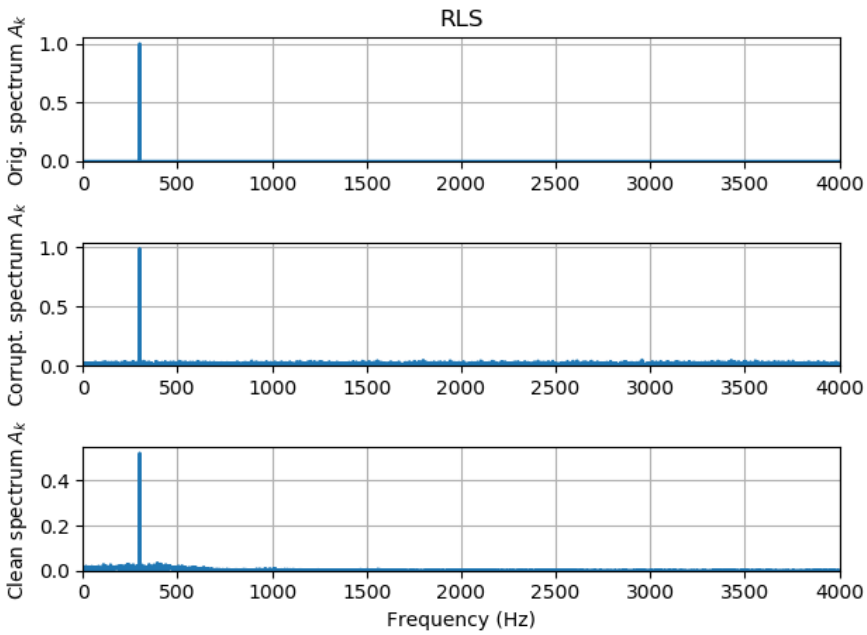
$$\left[w_0, \cdots, w_{N-1}\right] = [-0.140064, -0.139377, -0.144409, -0.106813, -0.071404, -0.061952, -0.043756, -0.015332, 0.002229, 0.799859, 0.019485, 0.037703, 0.0255, 0.04034, 0.069252, 0.096735]$$



The corrupted speech, reference noise, cleaned speech



Spectral of corrupted speech, cleaned speech



Problem 9.28 (Bonus)

Frequency tracking:

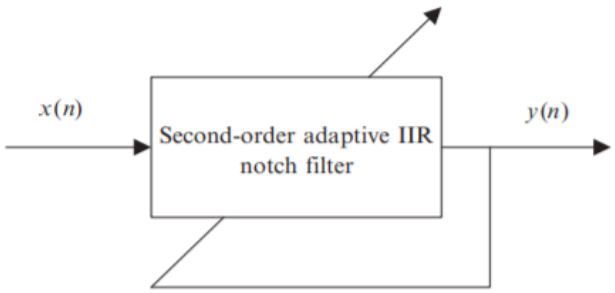


FIG. 9.30

A frequency tracking system.

An adaptive filter can be applied for real-time frequency tracking (estimation). In this application, a special second notch IIR filter structure, as shown in Fig. 9.30, is preferred for simplicity. The notch filter transfer function

$$H(z) = \frac{1 - 2\cos(\theta)z^{-1} + z^{-2}}{1 - 2r\cos(\theta)z^{-1} + r^2 z^{-2}}$$

has only one adaptive parameter θ. It has two zeros on the unit circle resulting in an infinite-depth notch. The parameter r controls the notch bandwidth. It requires 0 << r < 1 for achieving a narrowband notch. When r is close to 1, the 3-dB notch filter bandwidth can be approximated as BW = 2(1 - r) (see Chapter 8). The input sinusoid whose frequency f needs to be estimated and tracked is given below:

$$x(n) = A\cos(2\pi fn/f_s + \alpha)$$

where A and α are the amplitude and phase angle. The filter output is expressed as

$$y(n) = x(n) - 2\cos[\theta(n)]x(n-1) + x(n-2) + 2r\cos[\theta(n)]y(n-1) - r^2 y(n-2)$$

The objective is to minimize the filter instantaneous output power y^2(n). Once the output power is minimized, the filter parameter θ=2πf/f_s will converge to its corresponding frequency f(Hz).

The LMS algorithm to minimize the instantaneous output power y^2(n) is given as

$$\theta(n+1) = \theta(n) - 2\mu y(n)\beta(n)$$

where the gradient function β(n)=∂y(n)/∂θ(n) can be derived as follows:

$$\beta(n) = 2\sin[\theta(n)]x(n-1) - 2r\sin[\theta(n)]y(n-1) + 2r\cos[\theta(n)]\beta(n-1) - r^2\beta(n-2)$$

and μ is the convergence factor which controls the speed of algorithm convergence.

In this project

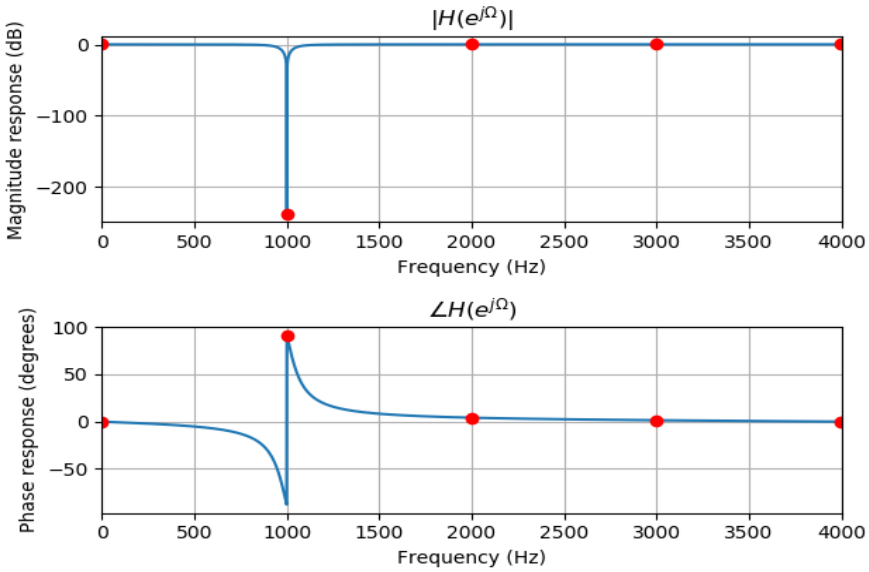
1. plot and verify the notch frequency response by setting f_s=8000 Hz, f=1000Hz, and r=0.95.
2. Then generate the sinusoid with duration of 10 s, frequency of 1000 Hz, and amplitude of 1.
3. Implement the adaptive algorithm using an initial guess θ(0)=2π ×2000/fs = 0.5π
4. plot the tracked frequency f(n)=θ(n) f_s/2π for tracking verification.

Notice that

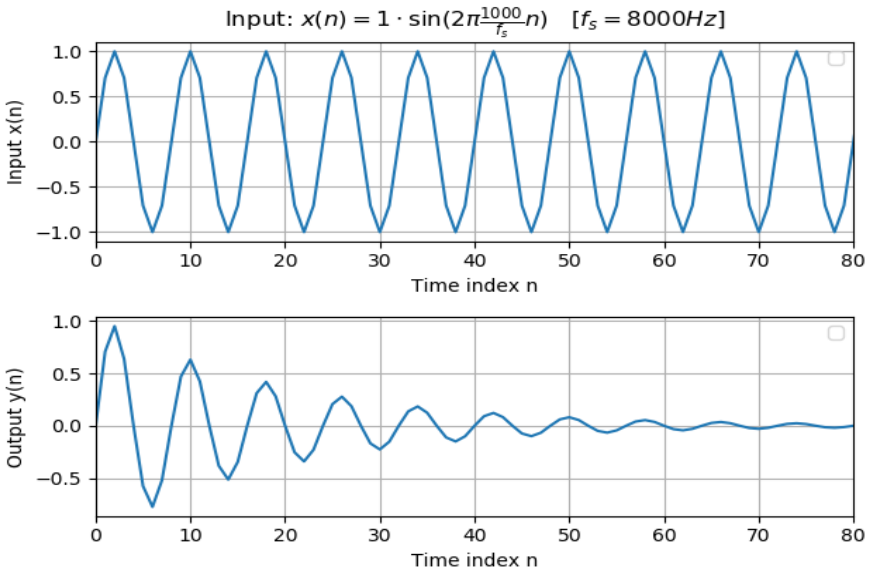
- this particular notch filter only works for a single frequency tracking, since the mean squared error function E[y^2(n)] has a one global minimum (one best solution when the LMS algorithm converges). Details of adaptive notch filter can be found in the reference (Tan and Jiang, 2012).
- the general IIR adaptive filter suffers from local minima, that is, the LMS algorithm converges to local minimum and the nonoptimal solution results in.

solution

1. plot and verify the notch frequency response by setting f_s=8000 Hz, f=1000Hz, and r=0.95.



2. Then generate the sinusoid with duration of 10 s, frequency of 1000 Hz, and amplitude of 1.




```
from math import sin
fs, f = 8000, 1000
N = 10 * fs # 10 second duration
list_x = [sin(2*pi * (f/fs) * ind) for ind in range(N)]
```

3. Implement the adaptive algorithm using an initial guess θ(0)=2π ×2000/fs = 0.5π

```
from math import cos, acos, pi
class Freq_track(object):
    def __init__(self, mu, r, theta_0=pi/2):
        self.mu = mu # learning rate
        self.r = r
        self.c = cos(theta_0)
        self.X = [0, 0, 0] # x(n), x(n-1), x(n-2)
        self.Y = [0, 0]    # y(n-1), y(n-2)
        self.Beta = [0, 0] # beta := dy/dc; beta(n-1), beta(n-2)
        # optional: coeffs for x(n), y(n)
        self.coef_x = [1, -2 * self.c, 1]
        self.coef_y = [2 * self.r * self.c , - self.r * self.r]
        self.coef_x_beta = [0, -2, 0]
        self.coef_y_beta = [2 * self.r, 0]
    def get_theta(self):
        return acos(self.c)
    def __update_X(self, x): # __func(): private method
        self.X = [x] + self.X[:-1] # update X(n)
    def __update_Y_Beta(self, y, beta):
        self.Y = [y] + self.Y[:-1]
        self.Beta = [beta] + self.Beta[:-1]
    def __clip_c(self):
        if self.c > 1:
            self.c = 1
        elif self.c < -1:
            self.c = -1
    def __update_coef(self):
        self.coef_x = [1, -2 * self.c, 1]
        self.coef_y = [2 * self.r * self.c , - self.r * self.r]
        self.coef_x_beta = [0, -2, 0]
        self.coef_y_beta = [2 * self.r, 0]
    def train(self, x):
        self.__update_X(x)
        # calc old y(n), beta(n)
        y_old =\
            sum([coef_ * x_ for coef_, x_ in list(zip(self.coef_x, self.X))])\
            +sum([coef_ * y_ for coef_, y_ in list(zip(self.coef_y, self.Y))])
        beta_old =\
            sum([coef_ * x_ for coef_, x_ in list(zip(self.coef_x_beta, self.X))])\
            +sum([coef_*y_ for coef_, y_ in list(zip(self.coef_y_beta, self.Y))])\
            +sum([coef_ * b_ for coef_, b_ in list(zip(self.coef_y, self.Beta))])
        # update self.c := cos(theta)
        self.c = self.c - 2 * self.mu * y_old * beta_old
        self.__clip_c()
        self.__update_coef()
        # calc new y(n), beta(n) with new self.c
        y =\
            sum([coef_ * x_ for coef_, x_ in list(zip(self.coef_x, self.X))])\
            +sum([coef_ * y_ for coef_, y_ in list(zip(self.coef_y, self.Y))])
        beta =\
            sum([coef_ * x_ for coef_, x_ in list(zip(self.coef_x_beta, self.X))])\
```

```
            +sum([coef_*y_ for coef_, y_ in list(zip(self.coef_y_beta, self.Y))])\
            +sum([coef_ * b_ for coef_, b_ in list(zip(self.coef_y, self.Beta))])
        self.__update_Y_Beta(y, beta)


if __name__ == "__main__":
    from math import sin
    import matplotlib.pyplot as plt
    fs, f = 8000, 1000
    N = 10 * fs # 10 second duration
    list_x = [sin(2*pi * (f/fs) * ind) for ind in range(N)]
    ########################################################
    mu, r, theta_0 = 7e-4, 0.95, pi/2
    freq_track = Freq_track(mu=mu, r=r, theta_0=theta_0)
    list_freq = []
    for x in list_x:
        freq_track.train(x)
        omega = freq_track.get_theta()
        list_freq.append( fs * omega / (2 * pi) )
    ########################################################
    fig = plt.figure()
    plt.plot(list_freq)
    plt.xlabel("Time index n")
    plt.ylabel("Frequency (Hz)")
    plt.title(r"Frequency tracking: $f=f_s \times \frac{\theta}{2\pi}$")
    ax = plt.gca()
    ax.set_xlim([0, N])
    plt.grid()
    plt.tight_layout()
    fig.savefig("../p9_28.png")
    plt.show()
    ########################################################
```

4. plot the tracked frequency f(n)=θ(n) f_s/2π for tracking verification.