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)

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

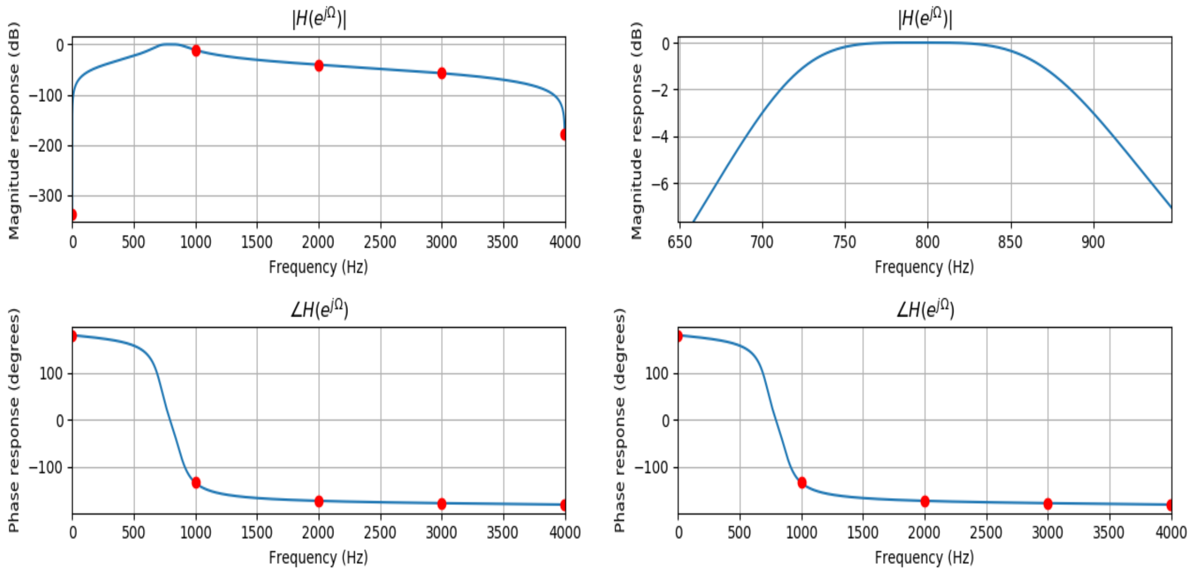
Substitute $s' = \frac{s^2+\omega_0^2}{sW}$ for band pass filter, thus

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

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

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

The magnitude and phase plots for the unknown system

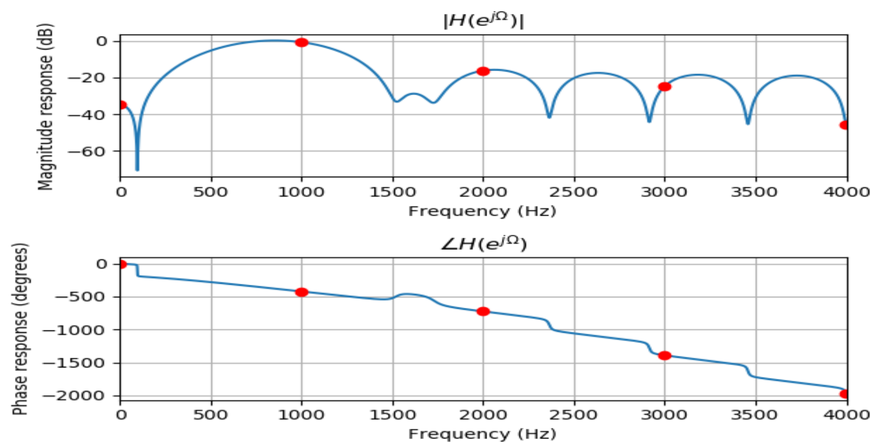


(c) Implement the adaptive FIR filter, plot the system input, system output, adaptive filter output, and the error signal, respectively.

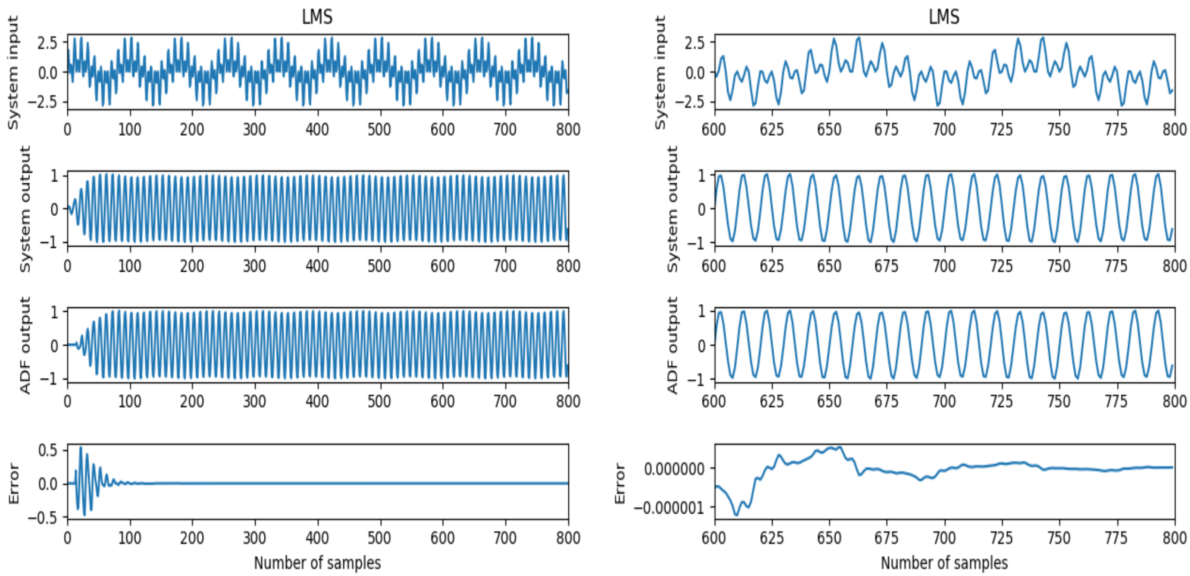
- Adaptive FIR filter
- Number of taps: 15 coefficients
- Algorithm: LMS algorithm
- Convergence factor: 0.01

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

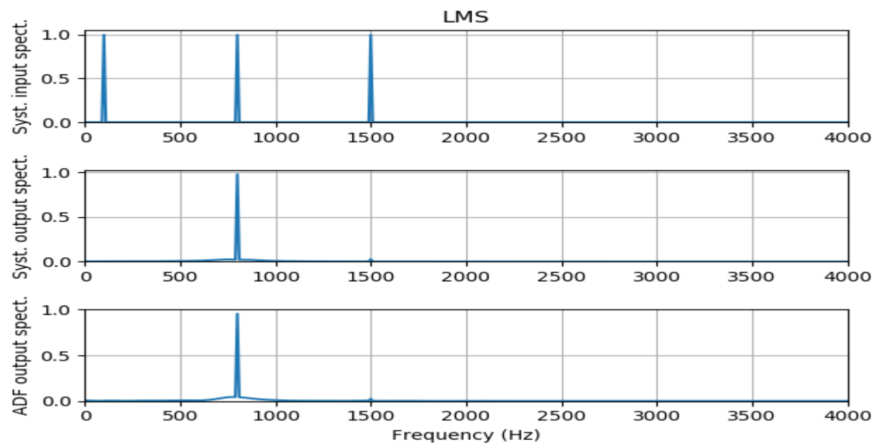
$$\left[w_0, \cdots, w_{N-1}\right] = [0.098584, 0.081271, 0.052333, -0.010238, -0.104861, -0.179288, -0.170288, -0.070568, 0.055421, 0.128914, 0.125045, 0.078709, 0.030333, -0.017251, -0.080168]$$



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



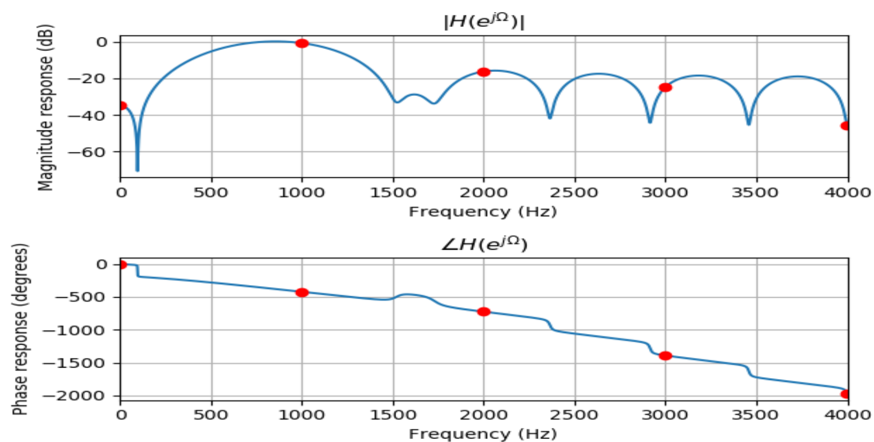
(d) Plot the input spectrum, system output spectrum, and adaptive filter output spectrum, respectively.



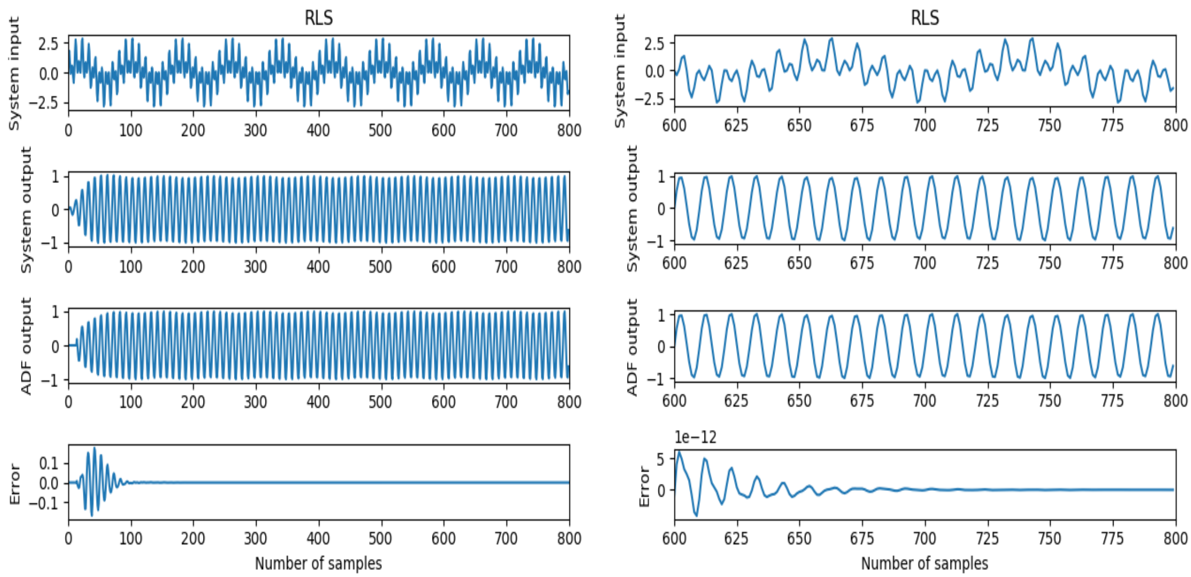
(e) Repeat (a)–(d) using the RLS algorithm with δ=1 and λ=0.96.

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

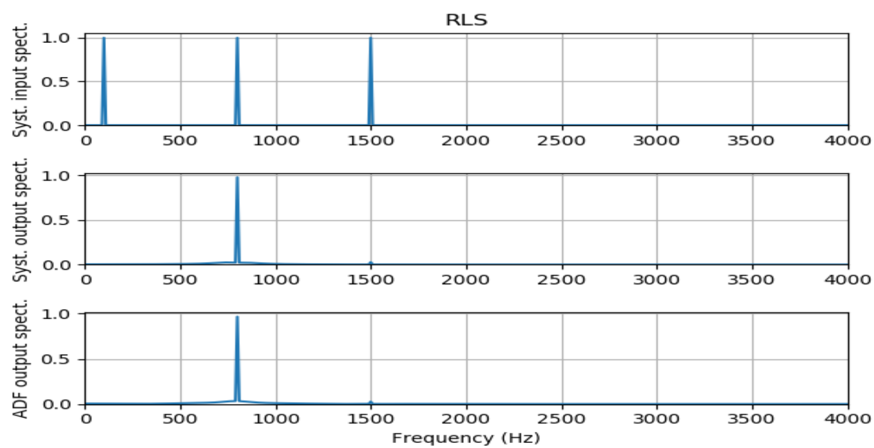
$$\left[w_0, \cdots, w_{N-1}\right] = [0.098584, 0.081271, 0.052333, -0.010238, -0.104861, -0.179288, -0.170288, -0.070568, 0.055421, 0.128914, 0.125045, 0.078709, 0.030333, -0.017251, -0.080168]$$



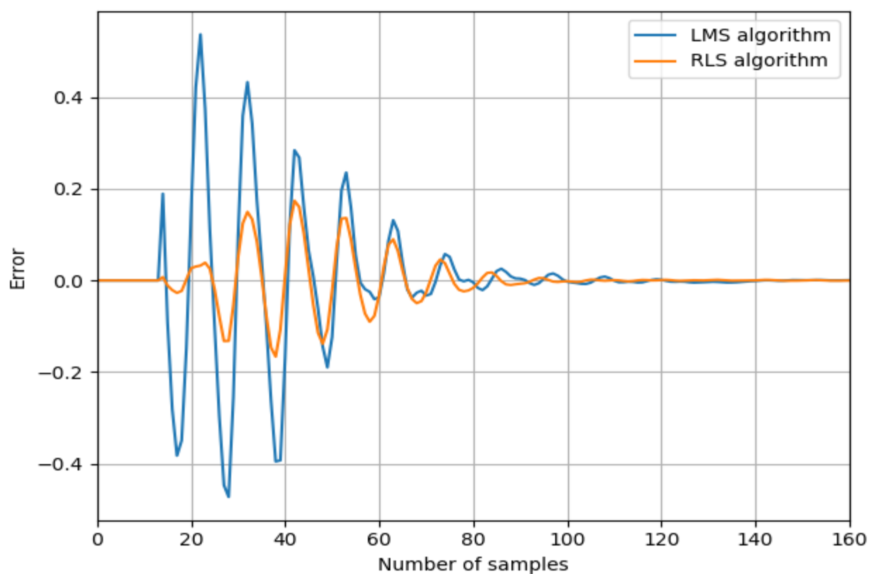
Plot of system input, system output, adaptive filter output, and the error signal.



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.

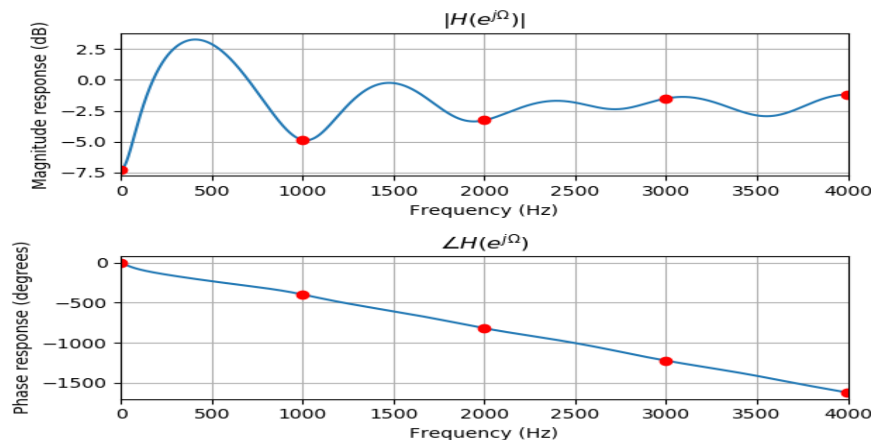
(c) Compare the spectral plots between the corrupted signal and the enhanced signal.

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

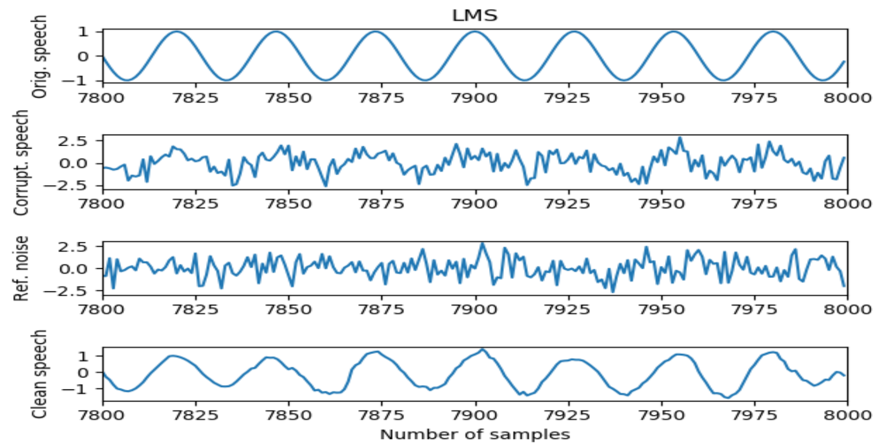
solution

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

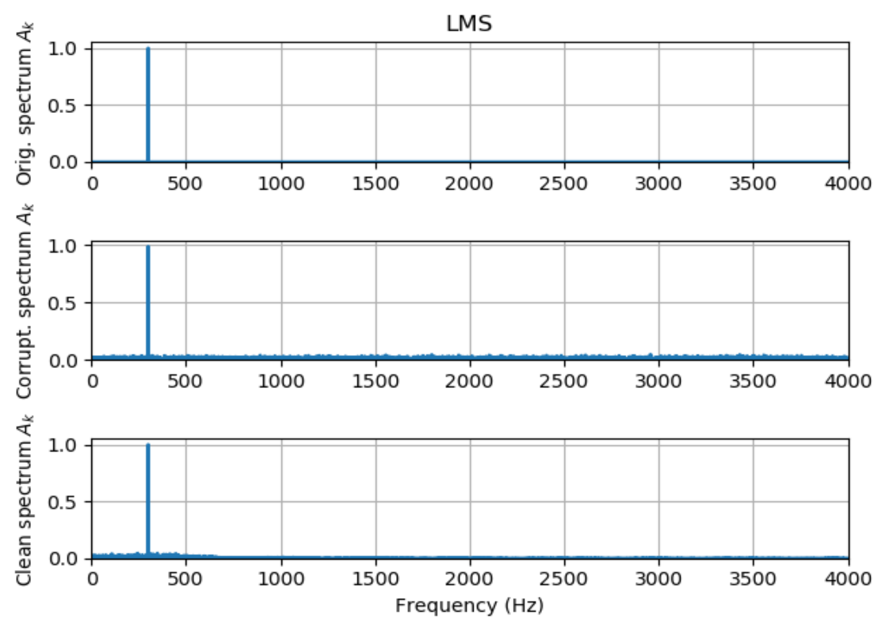
$$\left[w_0, \cdots, w_{N-1}\right] = [-0.117458, -0.120189, -0.116545, -0.118012, -0.09756, -0.087345, -0.067977, -0.046143, -0.012461, 0.803577, 0.041275, 0.067751, 0.068472, 0.070101, 0.082823, 0.083464]$$



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



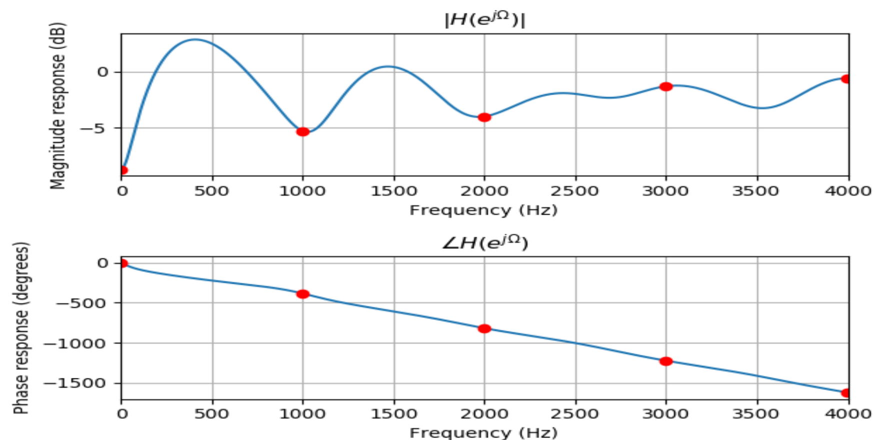
3. Compare the spectral plots between the corrupted signal and the enhanced signal.



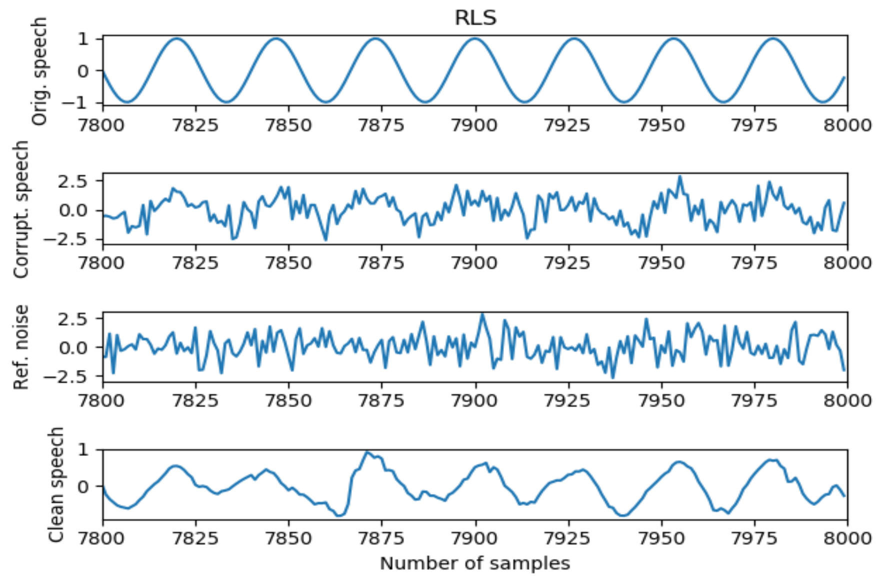
4. Repeat (a)–(c) using the RLS algorithm with δ=1 and λ=0.96.

The adaptive FIR filter

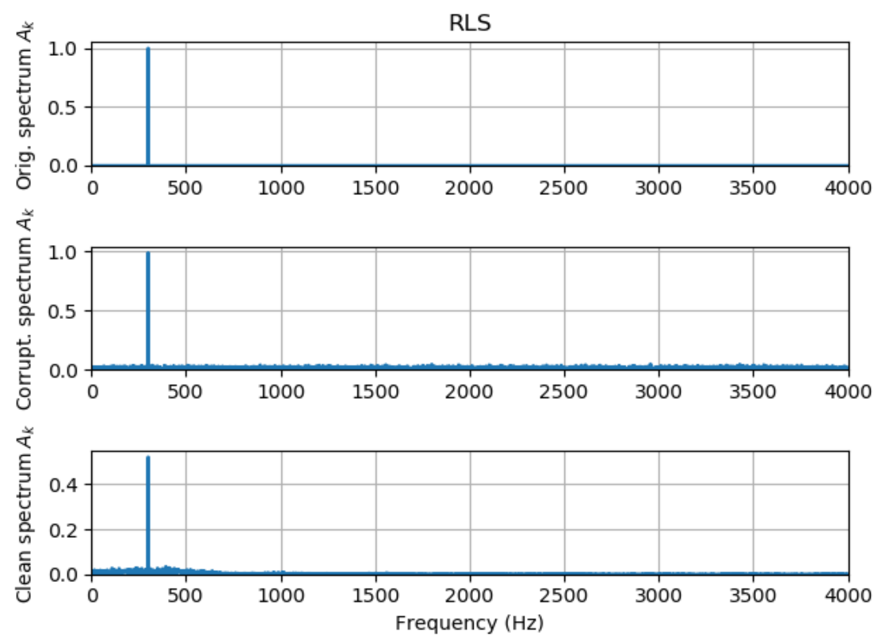
$$\left[w_0, \cdots, w_{N-1}\right] = [-0.140064, -0.139377, -0.144409, -0.106813, -0.071404, -0.061952, -0.043756, -0.015332, 0.002229, 0.799859, 0.019485, 0.037703, 0.0255, 0.04034, 0.069252, 0.096735]$$



The corrupted speech, reference noise, cleaned speech



Spectral of corrupted speech, cleaned speech



Problem 9.28 (Bonus)

Frequency tracking:

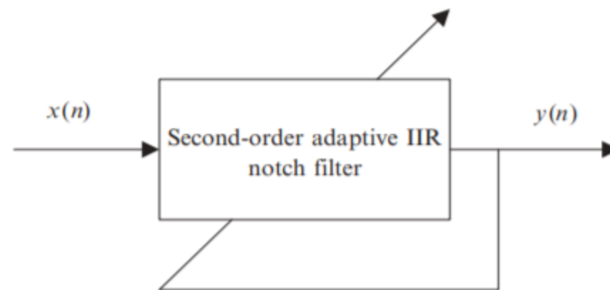


FIG. 9.30

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 $0 \ll r < 1$ 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

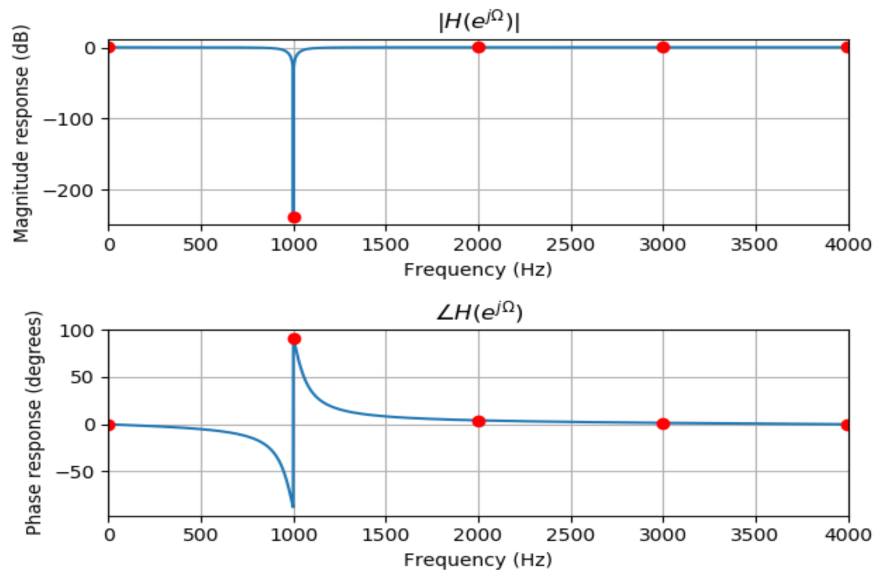
1. plot and verify the notch frequency response by setting f_s=8000 Hz, f=1000Hz, and r=0.95.
2. Then generate the sinusoid with duration of 10 s, frequency of 1000 Hz, and amplitude of 1.
3. Implement the adaptive algorithm using an initial guess θ(0)=2π $\times$2000/fs = 0.5π
4. plot the tracked frequency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

Notice that

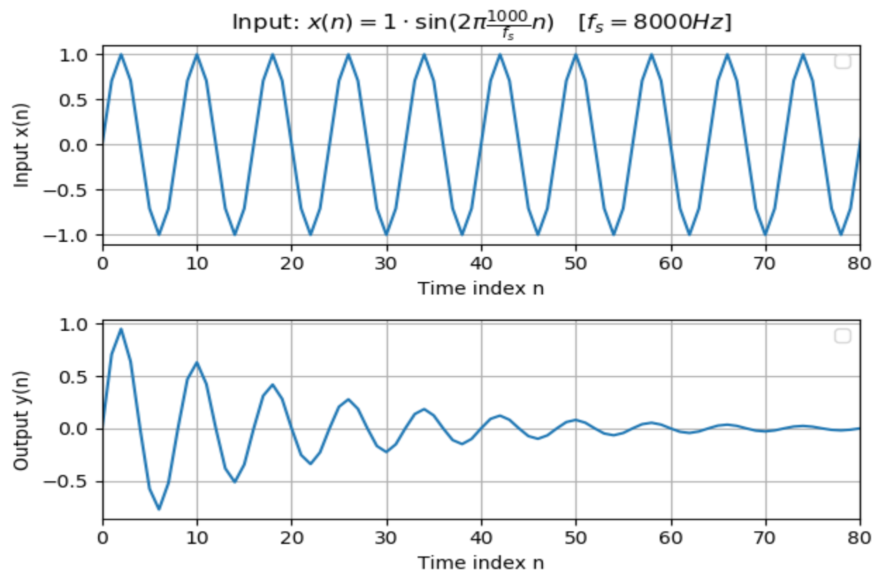
- this particular notch filter only works for a single frequency tracking, since the mean squared error function E[y^2(n)] has a one global minimum (one best solution when the LMS algorithm converges). Details of adaptive notch filter can be found in the reference (Tan and Jiang, 2012).
- the general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