

DSP Homework

Xu, Minhuan

December 11, 2022

Contents

1	Videos	1
2	How fast do Mosquitoes Flap	1
2.1	Frequency Spectrum	1
2.2	Counting With Schmidt Trigger	2
2.3	Results	2
3	Base-band Wireless Communication Simulate	2
3.1	Simulation and Stability Analysis	2
3.2	Unstable Behavior When Noise is Zero	3
A	Possible Mistakes in Lesson	3
B	Code Listing	5

Abstract

1 Videos

2 How fast do Mosquitoes Flap

2.1 Frequency Spectrum

First, draw the frequency spectrum, see Fig. 1 The fundamental frequency is about 400 Hz, if we want to know the

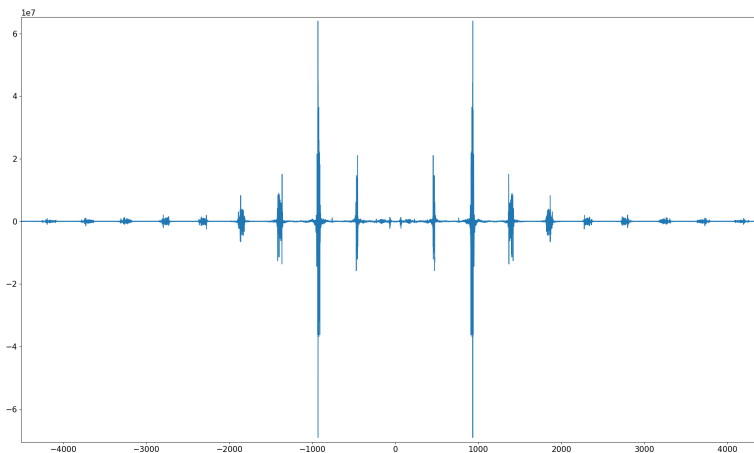


Figure 1: The Spectrum Of Mosquitoes Flipping Their Wings

instantaneous frequency mosquitoes flip, we only need the fundamental frequency, see Fig. 2.

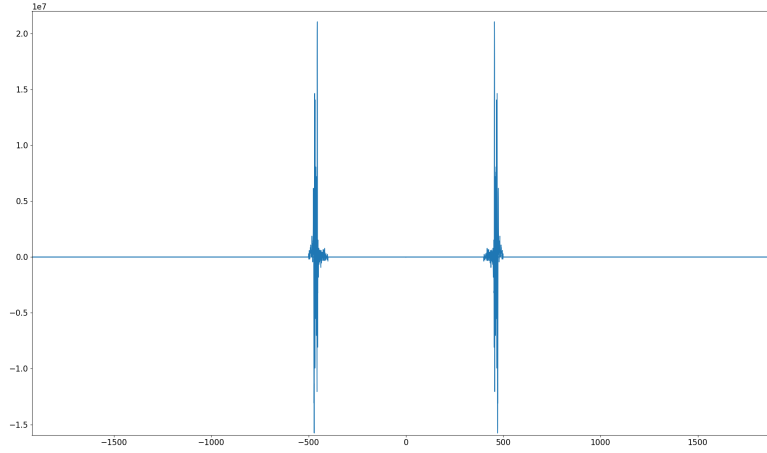


Figure 2: Base-band Spectrum of Mosquitoes' Flipping

2.2 Counting With Schmidt Trigger

To know the instantaneous frequency, I want to cut the wave file into small slice, count how many times the line cross the zero point, and divide this count with the duration time of that small slice to get the frequency here. We usually use Schmidt trigger to this counting job, see Fig. 3.

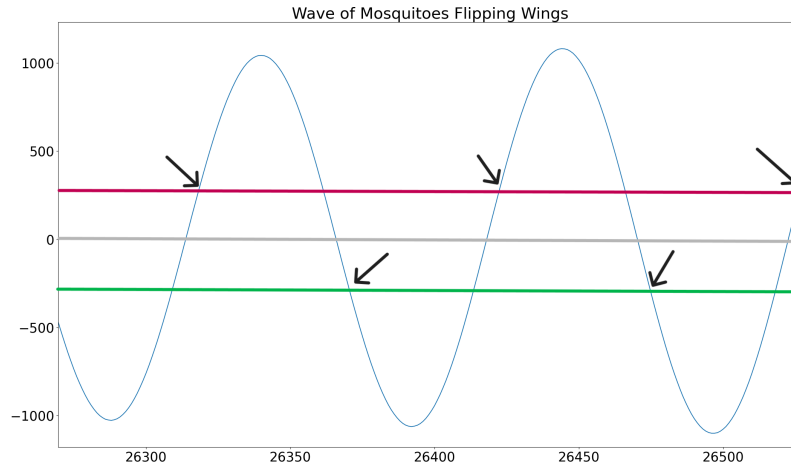


Figure 3: Schmidt Trigger Applied Here

2.3 Results

Here are my results, see Fig. 4.

3 Base-band Wireless Communication Simulate

3.1 Simulation and Stability Analysis

I wrote my code by imitating your code, but I made some changes because I thought there might be a problem with the code in class. This mistake will explain the unstable behavior when noise (along with calculation error) is zero. I wrote my comparison in appendix.

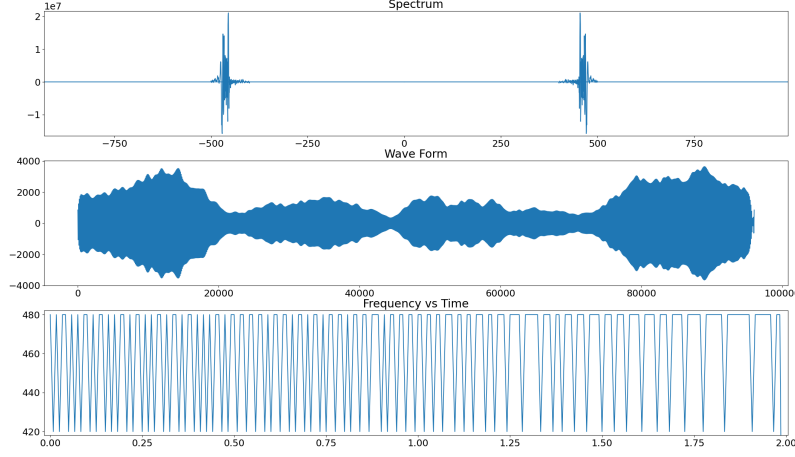


Figure 4: My Results

In recovery process, we have

$$\begin{aligned} y &= x * h + n \\ \tilde{y} &= \tilde{x} \times \tilde{h} + N_0 \end{aligned} \quad (1)$$

So, the recovered signal \hat{x} is as below

$$\begin{aligned} \mathcal{F}\{\hat{x}\} &= \tilde{y} \times \frac{1}{\tilde{h}} \\ &= (\tilde{x} \times \tilde{h} + N_0) \times \frac{1}{\tilde{h}} \\ &= \tilde{x} + \frac{N_0}{\tilde{h}} \end{aligned} \quad (2)$$

Here in (2), the IIR N_0/\tilde{h} , is why \hat{x} can be unstable.

3.2 Unstable Behavior When Noise is Zero

In mathematics, it is impossible to see unstable behavior in (3).

$$\begin{aligned} \mathcal{F}\{\hat{x}\} &= \tilde{y} \times \frac{1}{\tilde{h}} \\ &= (\tilde{x} \times \tilde{h}) \times \frac{1}{\tilde{h}} \\ &= \tilde{x} \end{aligned} \quad (3)$$

So, there must be other noise in the code. What I found is that there will be a very small error in division of python, and it is exactly this little error that makes $1/\tilde{h}$ to oscillate infinitely.

Please see the terminal output in Fig. 5 which proves my guess.

This error is revealed in float numbers like $2.99 \dots 96$ and $-4.00 \dots 02$ in line 4 and 6 of Fig. 5.

There are my comparison between whether there is noise or calculation error in signal y , see Fig. 6.

We can see in Fig. 6(a) that when \tilde{h} is unstable, \hat{x} recovered from y only with calculation error went infinite, but y without both calculation error and noise can be perfectly recovered. This phenomenon is because unstable amplified the noise or the calculation error in recovery, so if there's no disturb in y , it is possible to recover the original signal x .

Also, when \tilde{h} is stable or is decrement oscillation, the noise and the calculation error will be decreasing along with the recovery process, so without noise is no longer a necessary condition for signal recovery. However, in this condition, only y without any noise can be perfectly recovered ($\hat{x} = x$).

Appendix A Possible Mistakes in Lesson

Dear Yi, I may found a small mistake in your code on Tuesday's (Dec.6) lesson. It is about the recovering of from $y = x * h$.

```
(xmh) xmh@xmh-PC:~/DSP$ /home/xmh/miniconda3/envs/xmh/bin/python /home/xmh/DSP/
xh[ 1 ] -= h[ 1 ] * xh[ 0 ] , xh[ 0 ] = 1.0
xh[ 2 ] -= h[ 1 ] * xh[ 1 ] , xh[ 1 ] = -0.0
xh[ 2 ] -= h[ 2 ] * xh[ 0 ] , xh[ 0 ] = 1.0
xh[ 3 ] -= h[ 1 ] * xh[ 2 ] , xh[ 2 ] = 2.9999999999999996
xh[ 3 ] -= h[ 2 ] * xh[ 1 ] , xh[ 1 ] = -0.0
xh[ 4 ] -= h[ 1 ] * xh[ 3 ] , xh[ 3 ] = -4.0000000000000002
xh[ 4 ] -= h[ 2 ] * xh[ 2 ] , xh[ 2 ] = 2.9999999999999996
xh[ 5 ] -= h[ 1 ] * xh[ 4 ] , xh[ 4 ] = 2.99999999999999947
xh[ 5 ] -= h[ 2 ] * xh[ 3 ] , xh[ 3 ] = -4.0000000000000002
xh[ 6 ] -= h[ 1 ] * xh[ 5 ] , xh[ 5 ] = -1.4401005572126297e-14
xh[ 6 ] -= h[ 2 ] * xh[ 4 ] , xh[ 4 ] = 2.99999999999999947
xh[ 7 ] -= h[ 1 ] * xh[ 6 ] , xh[ 6 ] = -3.0000000000000004
xh[ 7 ] -= h[ 2 ] * xh[ 5 ] , xh[ 5 ] = -1.4401005572126297e-14
xh[ 8 ] -= h[ 1 ] * xh[ 7 ] , xh[ 7 ] = 3.999999999999999
xh[ 8 ] -= h[ 2 ] * xh[ 6 ] , xh[ 6 ] = -3.0000000000000004
xh[ 9 ] -= h[ 1 ] * xh[ 8 ] , xh[ 8 ] = 3.9999999999999996954
xh[ 9 ] -= h[ 2 ] * xh[ 7 ] , xh[ 7 ] = 3.999999999999999
```

Figure 5: Error in Division Calculation

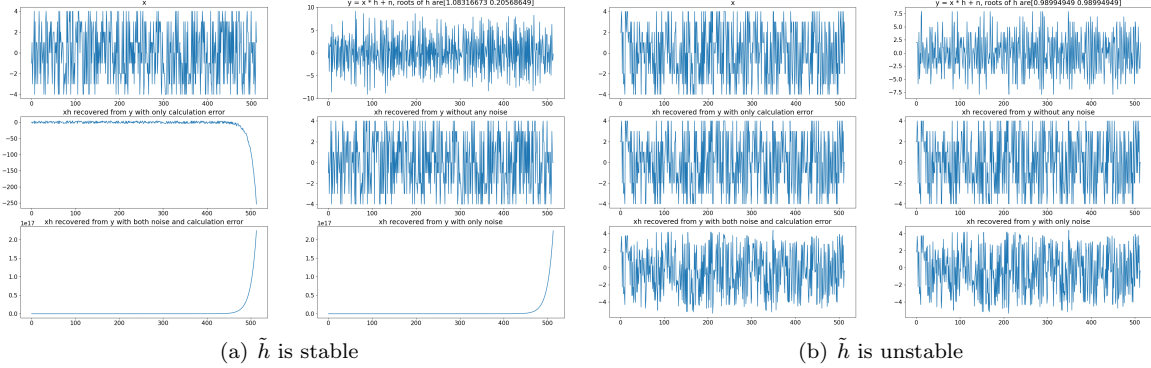


Figure 6: Stability Comparison when \tilde{h} is different

Please see the probably incorrect code as below:

```
def reX_estimate(y, h):
    N = len(y)
    M = len(h)
    print("Length of y is ", N, ", Length of h is ", M)
    xh = y.copy()
    for n in range(N):
        print("Info: n is now ", n)
        for m in range(1, min(M, n)): # Mistake here, n should be n + 1
            xh[n] -= h[m] * xh[n - m]
            print("xh[" , n, "] -= h[" , m, "] * xh[" , n - m, "]")
        xh[n] /= h[0]
    return xh
```

The length of h is 3 and the roots of h are about $[1.77, 0.57]$. Key point is in the output, see Fig. 7.

```
Length of y is 130 , Length of h is 3
Info: n is now 0
Info: n is now 1
Info: n is now 2
xh[ 2 ] -= h[ 1 ] * xh[ 1 ]
Info: n is now 3
xh[ 3 ] -= h[ 1 ] * xh[ 2 ]
xh[ 3 ] -= h[ 2 ] * xh[ 1 ]
Info: n is now 4
xh[ 4 ] -= h[ 1 ] * xh[ 3 ]
xh[ 4 ] -= h[ 2 ] * xh[ 2 ]
```

Figure 7: Output of Test Code

We can find that while $\hat{x}[2] = y[2] = h[0] \times x[2] + h[1] \times x[1] + h[2] \times x[0]$, this code only minus $h[1] \times x[1]$, that is the small mistake.

This small mistake in recovering x is *working as noise*, as a result, the $h(n)$ which is an unstable IIR, will *amplify this*

noise and finally cause the *unstable behavior*. As the Fig. 8 revealed.

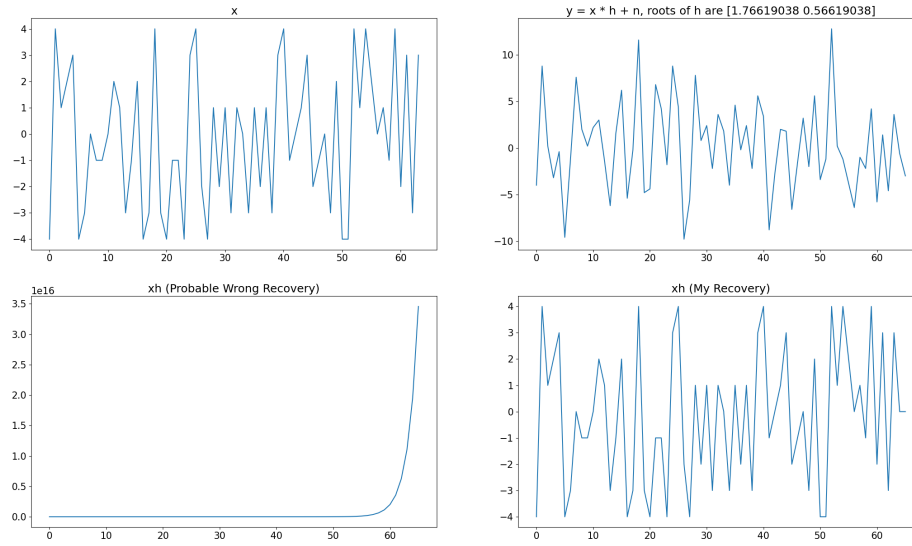


Figure 8: Different Behavior Between Previous Code and Current Code

Appendix B Code Listing

```
# simulate.py
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

def genPAM(N, M):
    """Generate PAM Signal"""
    return (np.random.randint(-M, M + 1, (N)))

def genNoise(y, snr = 20):
    """Generate White Noise"""
    Es = np.mean(y**2)
    sigma = np.sqrt(Es / 10 ** ( snr / 10))
    n = sigma * np.random.randn(len(y))
    return n + y, n

def reX_estimate(y, h, ErrorFlag=1):
    """Estimate x From y"""
    N = len(y)
    M = len(h)
    xh = y.copy()
    for n in range(N):
        # print("Info: n is now ", n)
        for m in range(1, min(M, n + 1)):
            xh[n] -= h[m] * xh[n - m]
            # if ErrorFlag and n <= 20:
            #     print("xh[" , n, "] -= h[" , m, "] * xh[" , n - m, "]", " , xh[" , n - m, "] =", xh[n - m])
        xh[n] /= h[0]
        if ErrorFlag == 0:
            xh[n] = np.round(xh[n], 8) #Exclude Error generated by calculation accuracy
    return xh

#parameters
N = 256
Nc = 3
M = 4
```

```

#generate x and h randomly
x = genPAM(N, M)
h = np.random.randn(Nc)
# h = [1, 0, -0.98]

#generate y as received signal
y = np.convolve(x, h)
yn, n = genNoise(y, snr = 30)

#recover x in different methods
xh = reX_estimate(y, h)
xh_new = reX_estimate(y, h, ErrorFlag=0)
xhn = reX_estimate(yn, h)
xhn_new = reX_estimate(yn, h, ErrorFlag=0)

#draw the results and the comparison
fig = plt.figure(figsize=(12, 8))

ax = fig.add_subplot(3, 2, 1)
ax.plot(x)
ax.set_title('x')
ax = fig.add_subplot(3, 2, 2)
ax.set_title('y = x * h + n, roots of h are' + str(np.abs(np.roots(h))))
ax.plot(y)
ax = fig.add_subplot(3, 2, 3)
ax.set_title('xh recovered from y with only calculation error')
ax.plot(xh)
ax = fig.add_subplot(3, 2, 4)
ax.set_title('xh recovered from y without any noise')
ax.plot(xh_new)
ax = fig.add_subplot(3, 2, 5)
ax.set_title('xh recovered from y with both noise and calculation error')
ax.plot(xhn)
ax = fig.add_subplot(3, 2, 6)
ax.set_title('xh recovered from y with only noise')
ax.plot(xhn_new)

plt.show()

```

```

# Mosquito.py
from scipy.io.wavfile import read, write
import numpy as np
from numpy.fft import fft, fftfreq, ifft
from matplotlib import pyplot as plt
plt.rcParams.update({'font.size': 18})

def loadWAV(fname):
    fs, w = read(fname)
    sec = len(w)/fs
    wt = fft(w)
    freq = fftfreq(wt.size, d=1/fs)
    """Simple High-Pass Filter"""
    fl = 400
    wt[int(-fl * sec):] = 0
    wt[:int(fl * sec)] = 0
    """Simple Low-Pass Filter"""
    fh = 500
    cfq = wt.size/2
    wt[int(cfq - (fs/2 - fh) * sec):int(cfq + (fs/2 - fh) * sec)] = 0
    return fs, w, wt, freq

def countFlip(w, fs, gate, intv):
    N = len(w)
    M = N//intv
    count = 0
    L = np.zeros(M)
    if w[0] > 0:
        flag = 1
    else:
        flag = -1
    for n in range(M):
        for m in range(intv):
            index = int(intv * n + m)
            if flag * w[index] > -gate:

```

```

        pass
    else:
        count += 1
        flag = w[index] / np.abs(w[index])
    L[n] = count/2
    count = 0
    return intv/fs, L

fs, w, wt, freq = loadWAV("/home/xmh/DSP/projF/code/mosquito-cut.wav")
iw = np.array(np.real(iff(wt)), dtype=np.int16)
write('/home/xmh/DSP/projF/code/mosquito-out.wav', fs, iw)

dt, L = countFlip(iw, fs, gate=300, intv=400)
F = L/(dt)
gapTime = np.arange(0, 2, dt)

fig = plt.figure(figsize=(12, 8))

ax = fig.add_subplot(311)
ax.set_title("Spectrum")
ax.plot(freq, np.real(wt))

ax = fig.add_subplot(312)
ax.set_title("Wave Form")
ax.plot(iw)

ax = fig.add_subplot(313)
ax.set_title("Frequency vs Time")
ax.plot(gapTime, F)

plt.show()

```