DSP Homework

Xu, Minhuan

December 10, 2022

Contents

Abstract

1 Videos

2 How fast do Mosquitoes Flap

pass

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.

In recovery process, we have

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

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

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

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).

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

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.

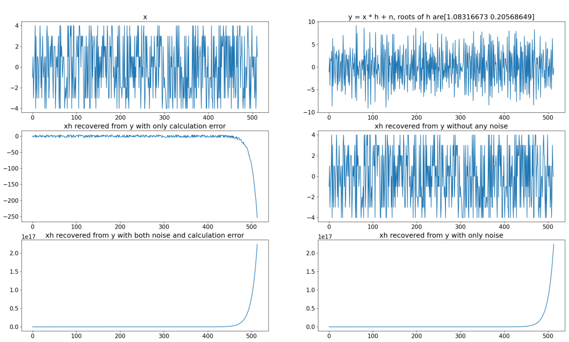
Please see the terminal output in Fig. 1 which proves my guess.

```
(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 ] = -4.000000000000002
xh[ 4 ] -= h[ 2 ] * xh[ 2 ] , xh[ 2 ] = 2.9999999999999996
xh[ 5 ] -= h[ 1 ] * xh[ 4 ] , xh[ 4 ] = 2.9999999999999947
xh[ 5 ] -= h[ 2 ] * xh[ 3 ] , xh[ 3 ] = -4.000000000000002
xh[ 6 ] -= h[ 1 ] * xh[ 5 ] , xh[ 5 ] = -1.4401005572126297e-14
xh[ 6 ] -= h[ 2 ] * xh[ 4 ] , xh[ 4 ] = 2.9999999999999947
xh[ 7 ] -= h[ 1 ] * xh[ 6 ] , xh[ 6 ] = -3.00000000000004
xh[ 7 ] -= h[ 2 ] * xh[ 5 ] , xh[ 5 ] = -1.4401005572126297e-14
xh[ 8 ] -= h[ 1 ] * xh[ 7 ] , xh[ 7 ] = 3.99999999999989
xh[ 8 ] -= h[ 2 ] * xh[ 6 ] , xh[ 6 ] = -3.00000000000004
xh[ 9 ] -= h[ 1 ] * xh[ 8 ] , xh[ 8 ] = 3.9999999999996954
xh[ 9 ] -= h[ 2 ] * xh[ 7 ] , xh[ 7 ] = 3.99999999999989
```

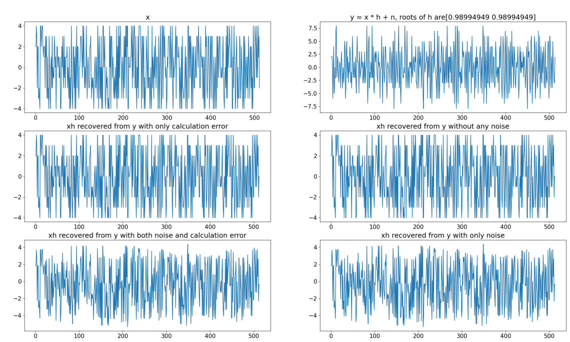
Figure 1: Error in Division Calculation

This error is revealed in float numbers like $2.99\cdots96$ and $-4.00\cdots02$ in line 4 and 6 of Fig. 1.

There are my comparison between whether there is noise or calculation error in signal y, see Fig. 2.



(a) $\tilde{h}$ is stable



(b) $\tilde{h}$ is unstable

Figure 2: Stability Comparison when $\tilde{h}$ is different

We can see in Fig. 2(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 Comparison Between Code in Class and My Code

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

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. 3.

```
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 3: 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. 4 revealed.

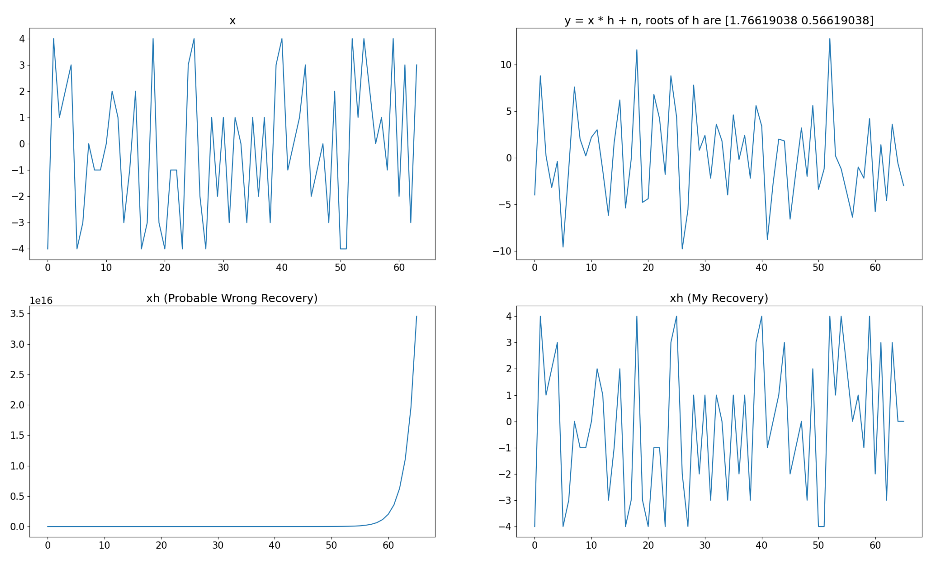


Figure 4: Different Behavior Between Previous Code and Current Code

Appendix B Code Listing

```
# simulate.py
import numpy as np
import matplotlib.pyplot as plt
```

```python
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()
```