REPORT

Zajęcia: Windowing

Teacher: prof. dr hab. Vasyl Martsenyuk

Lab 4

Date: 01.03.2024

Topic: "Influence of random signals on LTI systems"

Variant: 13

Agnieszka Białecka

Informatyka II stopień,

stacjonarne,

2 semestr,

Gr.1a

1. Abstract

Study a reaction of random signals on LTI systems with the help of numerical characteristics.

2. Theoretical introduction

2.1. Cross Correlation

The cross correlation function (CCF) for two (complex-valued) signals *x[k]* and *y[k]* of finite length *N* may be defined as

$$\varphi_{xy}[\kappa] = \frac{1}{N}\sum_{k=0}^{N-1} x[k+\kappa]\cdot y^*[k] = \frac{1}{N}\sum_{k=0}^{N-1} x[k]\cdot y^*[k-\kappa],$$

following the notation convention of the lecture and denoting $*$ as complex-conjugate. A very important property is

$$\varphi_{xy}[\kappa] = \varphi_{yx}^*[-\kappa].$$

Thus, only if the CCF is real-valued - which is the case if *x[k]* and *y[k]* are real valued - the equality $\varphi_{xy}[k] = \varphi_{yx}[-k]$ holds. Otherwise $\varphi_{xy}[k]$ and $\varphi_{yx}[-k]$ are linked by time-mirroring **and** complex-conjugate.

2.2. Cross Power Spectral Density

The cross power spectral density (CPSD) is the DTFT of the CCF, given as

$$\Phi_{xy}(\mathrm{e}^{\mathrm{j}\Omega}) = \mathrm{DTFT}\{\varphi_{xy}[\kappa]\} = \sum_{\kappa=-\infty}^{\infty} \varphi_{xy}[\kappa]\cdot \mathrm{e}^{-\mathrm{j}\Omega\kappa}.$$

Consequently, the inverse DTFT links the (cross)-PSD with the CCF as

$$\varphi_{xy}[\kappa] = \mathrm{IDTFT}\{\Phi_{xy}(\mathrm{e}^{\mathrm{j}\Omega})\} = \frac{1}{2\pi}\int_{-\pi}^{\pi} \Phi_{xy}(\mathrm{e}^{\mathrm{j}\Omega})\cdot \mathrm{e}^{+\mathrm{j}\Omega\kappa}\mathrm{d}\Omega.$$

2.3. Auto Correlation and Auto Power Spectral Density

If x and y are the same signal, then so called auto correlation function (ACF) $\varphi_{xy}[k]$ and (auto)-PSD $\varphi_{xx}(e^{j\Omega})$ are evaluated. These are defined and linked as

$$\varphi_{xx}[\kappa] = \frac{1}{N}\sum_{k=0}^{N-1} x[k+\kappa]\cdot x^*[k] = \frac{1}{N}\sum_{k=0}^{N-1} x[k]\cdot x^*[k-\kappa]$$

$$\varphi_{xx}[\kappa] = \mathrm{IDTFT}\{\Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega})\} = \frac{1}{2\pi}\int_{-\pi}^{\pi} \Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega})\cdot \mathrm{e}^{+\mathrm{j}\Omega\kappa}\mathrm{d}\Omega$$

$$\Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega}) = \mathrm{DTFT}\{\varphi_{xx}[\kappa]\} = \sum_{\kappa=-\infty}^{\infty} \varphi_{xx}[\kappa]\cdot \mathrm{e}^{-\mathrm{j}\Omega\kappa}$$

The following link

$$\varphi_{xx}[\kappa] = \varphi_{xx}^*[-\kappa]$$

with respect to time mirroring and complex-conjugate holds, since it is a special case of $\varphi_{xy}[k] = \varphi_{yx}^*[-k]$ for $x = y$. For $x \in R$, an additional important property can be deduced from

$$\varphi_{xx}[\kappa] \leq \varphi_{xx}[\kappa = 0] = \mathrm{E}\{x[k]\cdot x[k]\} = \sigma_x^2 + \mu_x^2.$$

This means, that evaluation of the ACF at $\kappa = 0$ results in the quadratic mean, i.e. the mean power. For all other values of κ, the ACF can only be less or equal than the mean power. Due to the IDTFT definition, also the relation

$$\varphi_{xx}[\kappa = 0] = \frac{1}{2\pi}\int_{-\pi}^{\pi} \Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega})\mathrm{d}\Omega$$

holds for the mean power, i.e. it is proportional to the area of the power spectral density.

2.4. Reaction of an LTI System on Random Signals

The relation between an input signal *x[k]*, the impulse response *h[k]* of an LTI system, and output signal *y[k]*, as well as the relation between their DTFT spectra $X(\mathrm{e}^{\mathrm{j}\Omega})$, $H(\mathrm{e}^{\mathrm{j}\Omega})$ and $Y(\mathrm{e}^{\mathrm{j}\Omega})$ are the fundamentals of DSP:

$$y[k] = x[k] * h[k] \circ\!-\!\bullet\, Y(\mathrm{e}^{\mathrm{j}\Omega}) = X(\mathrm{e}^{\mathrm{j}\Omega})\cdot H(\mathrm{e}^{\mathrm{j}\Omega}).$$

Similarly, the following relations can be stated for an input signal, drawn from a stationary random process, acting on an LTI system:

$$\mathrm{I}: \quad \varphi_{yx}[\kappa] = \varphi_{xx}[\kappa] * h[\kappa] \circ\!-\!\bullet\, \Phi_{yx}(\mathrm{e}^{\mathrm{j}\Omega}) = \Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega})\cdot H(\mathrm{e}^{\mathrm{j}\Omega}).$$

Interchanging of indices allows to write

$$\varphi_{xy}[\kappa] = \varphi_{xx}[\kappa] * h^*[-\kappa] \circ\!-\!\bullet\, \Phi_{xy}(\mathrm{e}^{\mathrm{j}\Omega}) = \Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega}) \cdot H^*(\mathrm{e}^{\mathrm{j}\Omega}).$$

The general rule for PSDs with *a, b* either *x* or *y* & *A, B* either *X* or *Y*

$$\Phi_{ab}(\mathrm{e}^{\mathrm{j}\Omega}) = A(\mathrm{e}^{\mathrm{j}\Omega}) \cdot B^*(\mathrm{e}^{\mathrm{j}\Omega}),$$

and the fact that

$$\Phi_{aa}(\mathrm{e}^{\mathrm{j}\Omega}) = A(\mathrm{e}^{\mathrm{j}\Omega}) \cdot A^*(\mathrm{e}^{\mathrm{j}\Omega}) = |A(\mathrm{e}^{\mathrm{j}\Omega})|^2,$$

and the further useful relations (actually from these two everything else can be deduced)

$$\begin{aligned} \mathrm{II}: \quad \Phi_{ya}(\mathrm{e}^{\mathrm{j}\Omega}) &= \Phi_{xa}(\mathrm{e}^{\mathrm{j}\Omega}) H(\mathrm{e}^{\mathrm{j}\Omega}) \\ \mathrm{III}: \quad \Phi_{ay}(\mathrm{e}^{\mathrm{j}\Omega}) &= \Phi_{ax}(\mathrm{e}^{\mathrm{j}\Omega}) H^*(\mathrm{e}^{\mathrm{j}\Omega}). \end{aligned}$$

are important to show the links between the functions.
The Wiener-Lee theorem links the input, the system response and output ACFs as

$$\mathrm{IV}: \quad \varphi_{yy}[\kappa] = \varphi_{xx}[\kappa] * \varphi_{hh}[\kappa] \circ\!-\!\bullet\, \Phi_{yy}(\mathrm{e}^{\mathrm{j}\Omega}) = \Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega}) \cdot |H(\mathrm{e}^{\mathrm{j}\Omega})|^2,$$

which can be also written as

$$\varphi_{yy}[\kappa] = \varphi_{xx}[\kappa] * \varphi_{hh}[\kappa] \circ\!-\!\bullet\, |Y(\mathrm{e}^{\mathrm{j}\Omega})|^2 = |X(\mathrm{e}^{\mathrm{j}\Omega})|^2 \cdot |H(\mathrm{e}^{\mathrm{j}\Omega})|^2.$$

The auto correlation function

$$\varphi_{hh}[\kappa] = h[\kappa] * h^*[-\kappa]$$

is called *filter ACF*.

Equations I, II, III and IV describe the fundamentals of random signal processing with LTI systems.

3. Input data (Variant)

This report was created with base of variant 13:
$\Omega_c = \pi/14$

GitHub repository:
https://github.com/Delisolara/AaDEC

4. Course of actions

4.1. Importing Libraries

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import signal
```

Picture 1. Uploaded libraries

4.2. Auto Correlation for Sine Signal

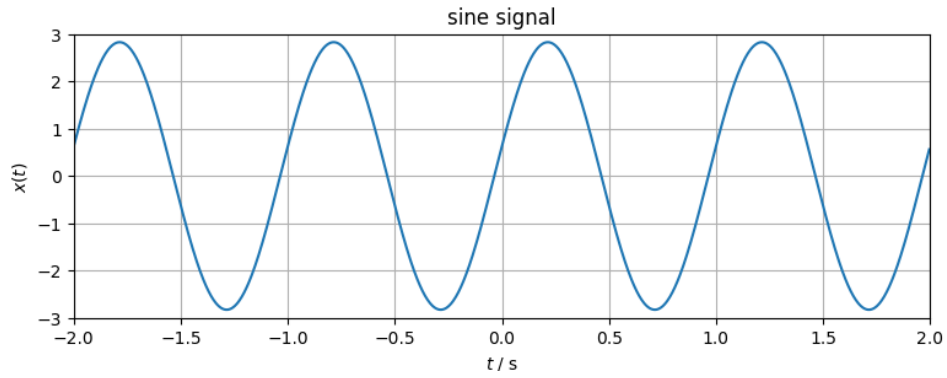
```
def my_xcorr2(x, y, scaleopt='none'):
    N = len(x)
    M = len(y)
    kappa = np.arange(0, N+M-1) - (M-1)
    ccf = signal.correlate(x, y, mode='full', method='auto')
    if N == M:
        if scaleopt == 'none' or scaleopt == 'raw':
            ccf /= 1
        elif scaleopt == 'biased' or scaleopt == 'bias':
            ccf /= N
        elif scaleopt == 'unbiased' or scaleopt == 'unbias':
            ccf /= (N - np.abs(kappa))
        elif scaleopt == 'coeff' or scaleopt == 'normalized':
            ccf /= np.sqrt(np.sum(x**2) * np.sum(y**2))
        else:
            print('scaleopt unknown: we leave output unnormalized')
    return kappa, ccf
```

Picture 2. Implemented code

```
w = 2*np.pi*1  # f = 1 Hz
tend = 4  # theoretically the sine has infinite duration, here only 4s to plot
N = tend * 2**8  # 256 samples per period -> very sufficient oversampling
t = np.arange(N)/N * tend - tend//2  # here: [-2s...+2s)
A = np.sqrt(8)  # this choice matches an ACF amplitude of 4
phi = np.pi/14  # arbitrary choice, note that ACF is not affected by phase
x = A*np.sin(w*t + phi)  # create the sine signal

# estimate the sampling frequency from the time interval between two samples
fs = 1 / (t[1]-t[0])
print('fs = {0:f} Hz'.format(fs))
plt.figure(figsize=(9, 3))
plt.plot(t, x)
plt.xlim(-2, 2)
plt.ylim(-3, 3)
plt.xlabel(r'$t$ / s')
plt.ylabel(r'$x(t)$')
plt.title('sine signal')
plt.grid(True)
```

Picture 3. Implemented code

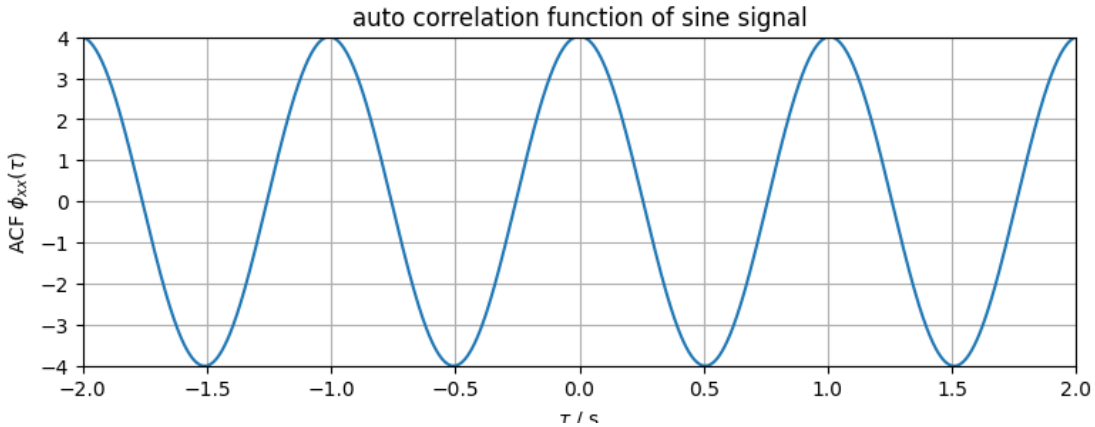
```
fs = 256.000000 Hz
```



Picture 4. The result

```
kappa, phixx = my_xcorr2(x, x, 'unbiased')

plt.figure(figsize=(9, 3))
plt.plot(kappa/fs, phixx)
plt.xlim(-2, 2)
plt.ylim(-4, 4)
plt.xlabel(r'$\tau$ / s')
plt.ylabel(r'ACF $\phi_{xx}(\tau)$')
plt.title('auto correlation function of sine signal')
plt.grid(True)
```

Picture 5. Implemented code



Picture 6. The result

4.3. CCFs

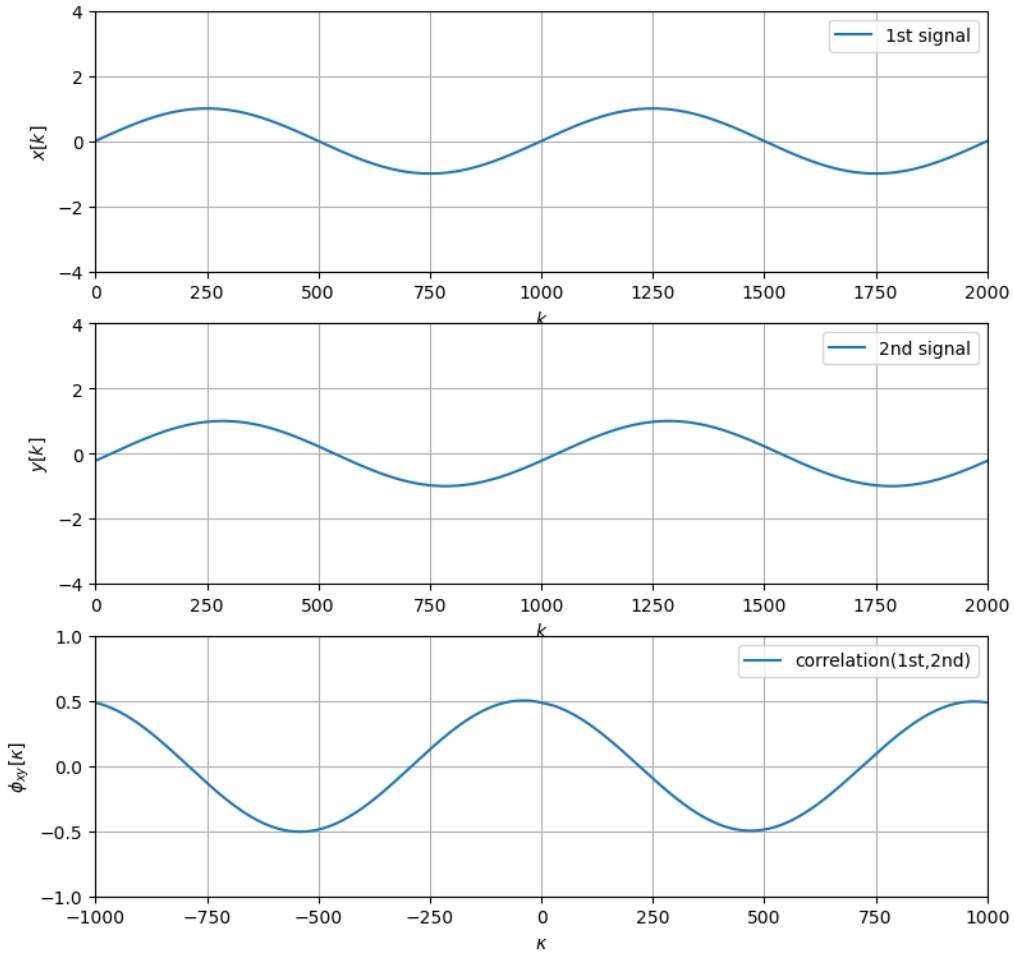
```
def my_ccf_plot(x, y, scaleopt):
    kappa, ccf = my_xcorr2(x, y, scaleopt)
    plt.figure(figsize=(9, 9))

    plt.subplot(3, 1, 1)
    plt.plot(x, label='1st signal')
    plt.xlim(0, 2000)
    plt.ylim(-4, 4)
    plt.xlabel(r'$k$')
    plt.ylabel(r'$x[k]$')
    plt.legend()
    plt.grid(True)

    plt.subplot(3, 1, 2)
    plt.plot(y, label='2nd signal')
    plt.xlim(0, 2000)
    plt.ylim(-4, 4)
    plt.xlabel(r'$k$')
    plt.ylabel(r'$y[k]$')
    plt.legend()
    plt.grid(True)

    plt.subplot(3, 1, 3)
    plt.plot(kappa, ccf, label='correlation(1st,2nd)')
    plt.xlim(-1000, 1000)
    plt.ylim(-1, 1)
    plt.xlabel(r'$\kappa$')
    plt.ylabel(r'$\phi_{xy}[\kappa]$')
    plt.legend()
    plt.grid(True)
```

Picture 7. Implemented code

```
N = 5000
k = np.arange(N)
Omega1 = 5 * 2*np.pi/N
Omega2 = 20 * 2*np.pi/N
```

```
# a)
x = np.sin(Omega1*k)
# b)
y = np.sin(Omega1*k - np.pi/14)
# c) and d)
my_ccf_plot(x, y, scaleopt='unbiased')
```

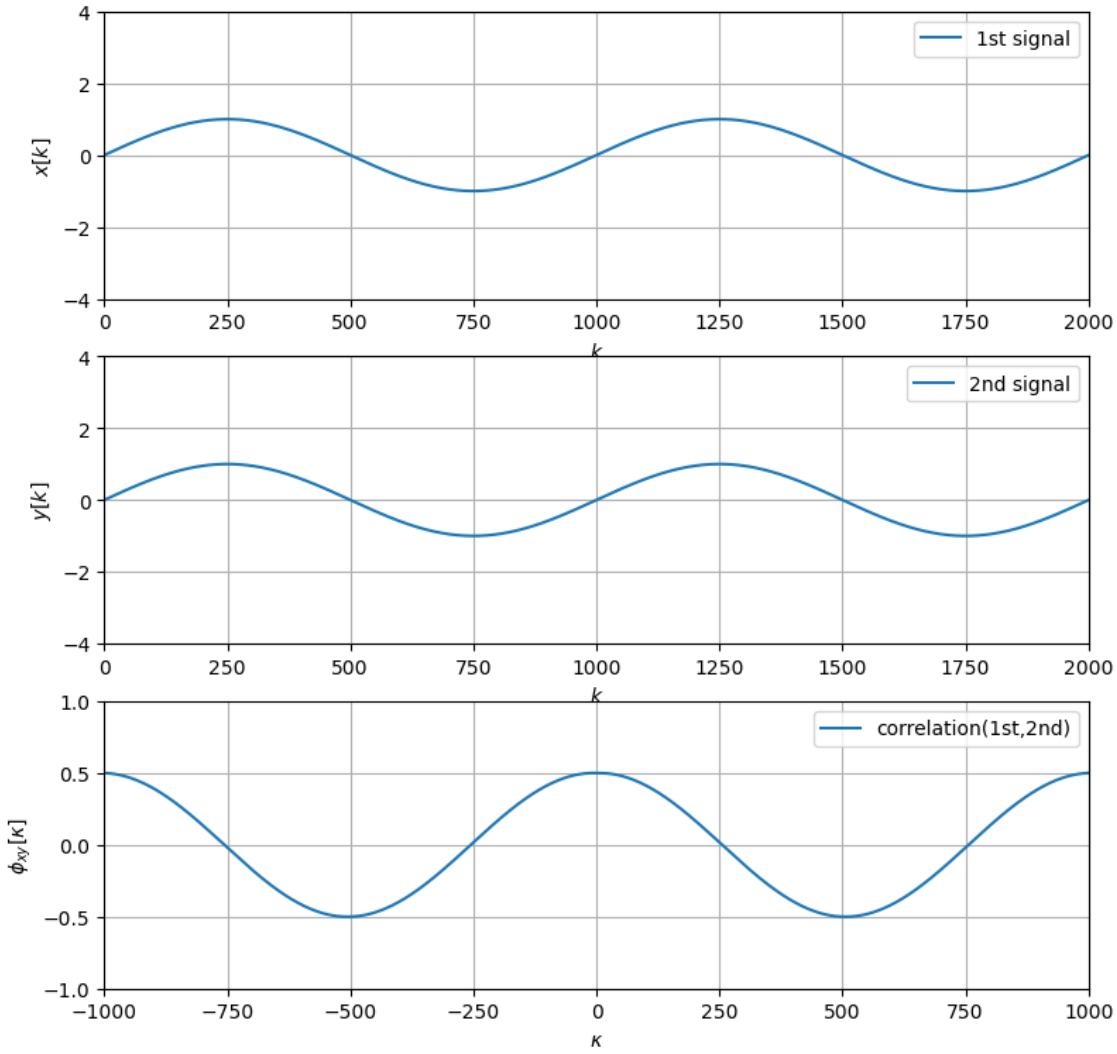
Picture 8. Implemented code



Picture 9. The result

```
# f)
x = np.sin(Omega1*k)
y = x
my_ccf_plot(x, y, scaleopt='unbiased')
```

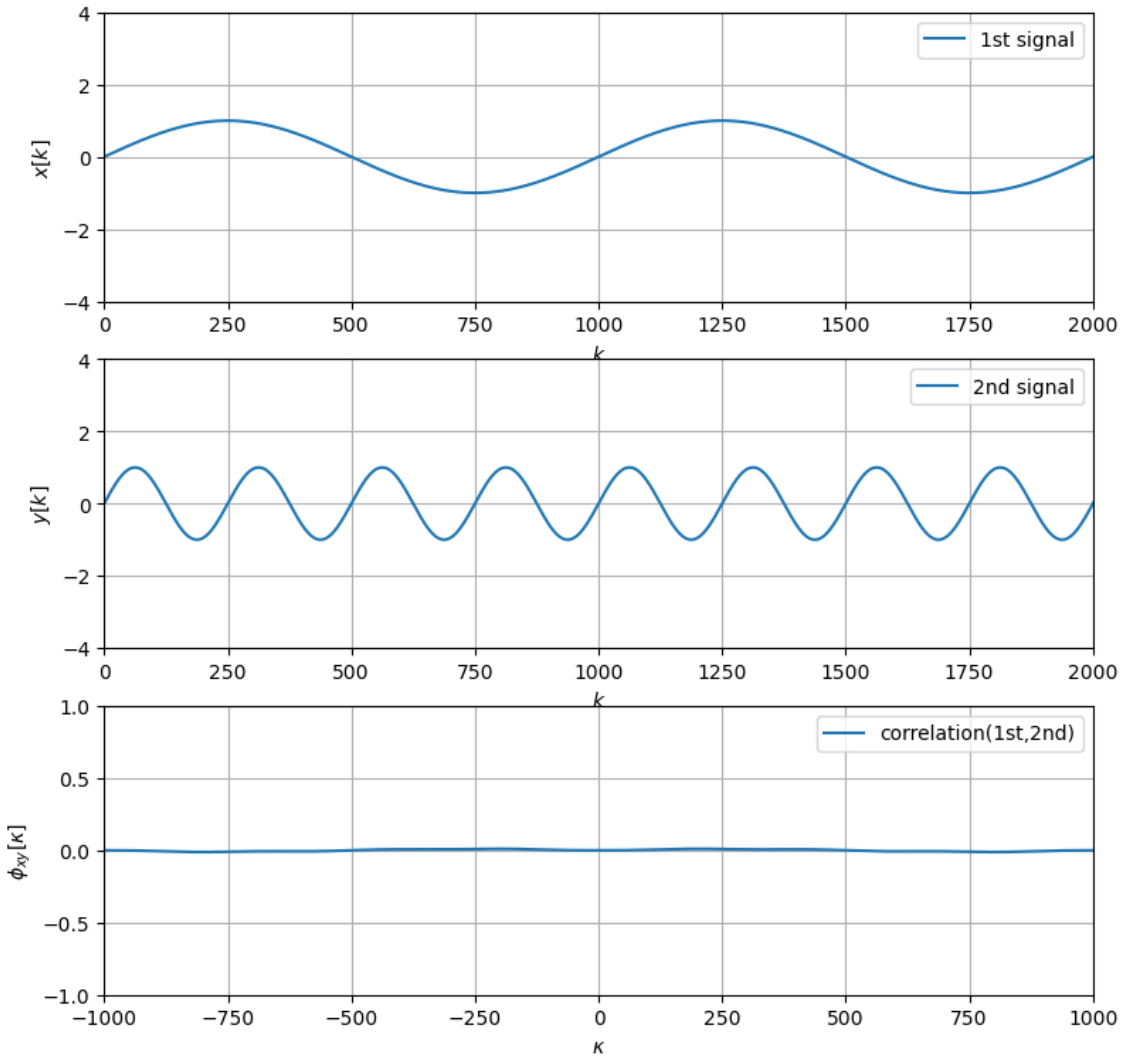
Picture 10. Implemented code



Picture 11. The result

```
# g)
x = np.sin(Omega1*k)
y = np.sin(Omega2*k)
my_ccf_plot(x, y, scaleopt='unbiased')
```

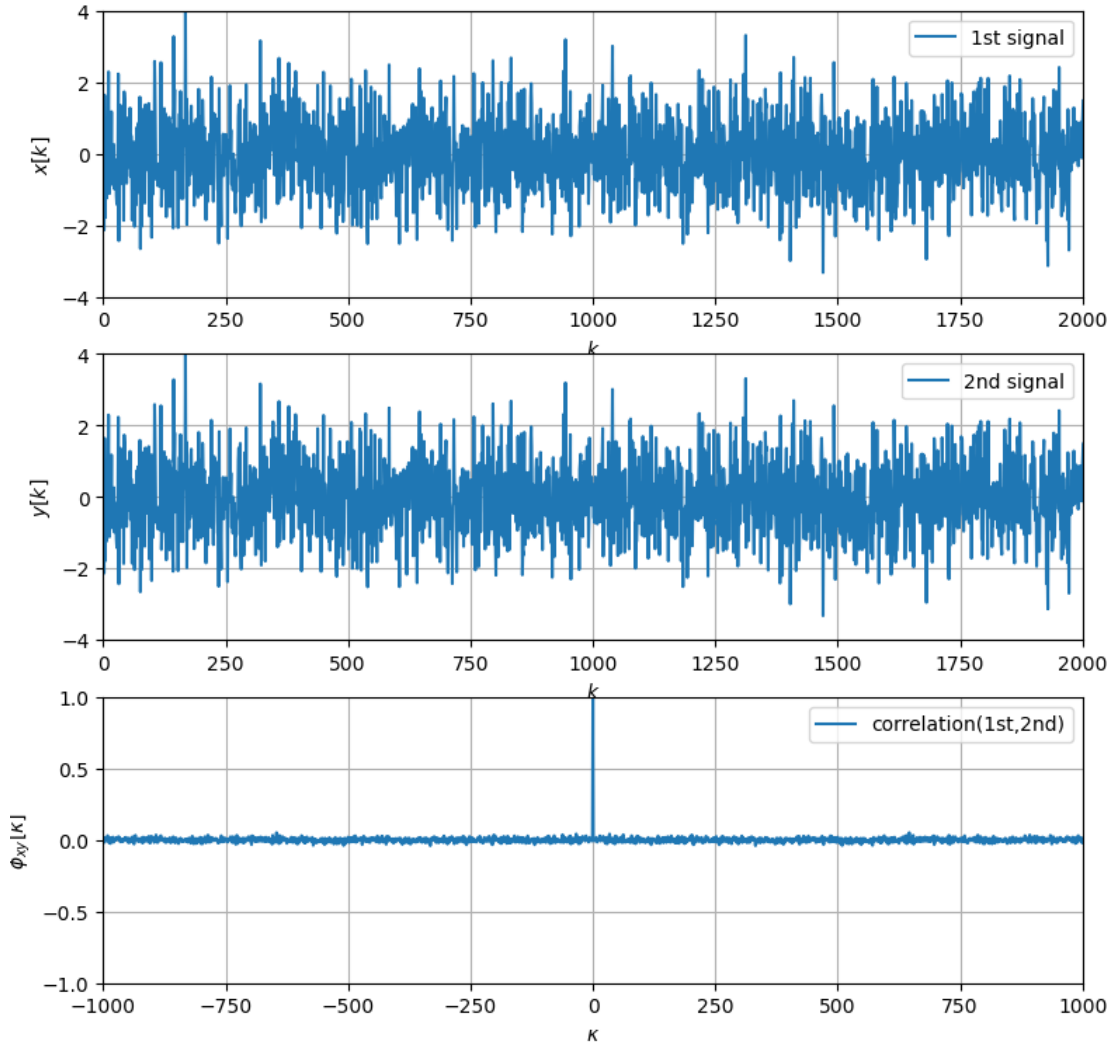
Picture 12. Implemented code



Picture 13. The result

```
# h)
np.random.seed(2)  # arbitrary choice
x = np.random.randn(N)
y = x
my_ccf_plot(x, y, scaleopt='biased')
```

Picture 14. Implemented code



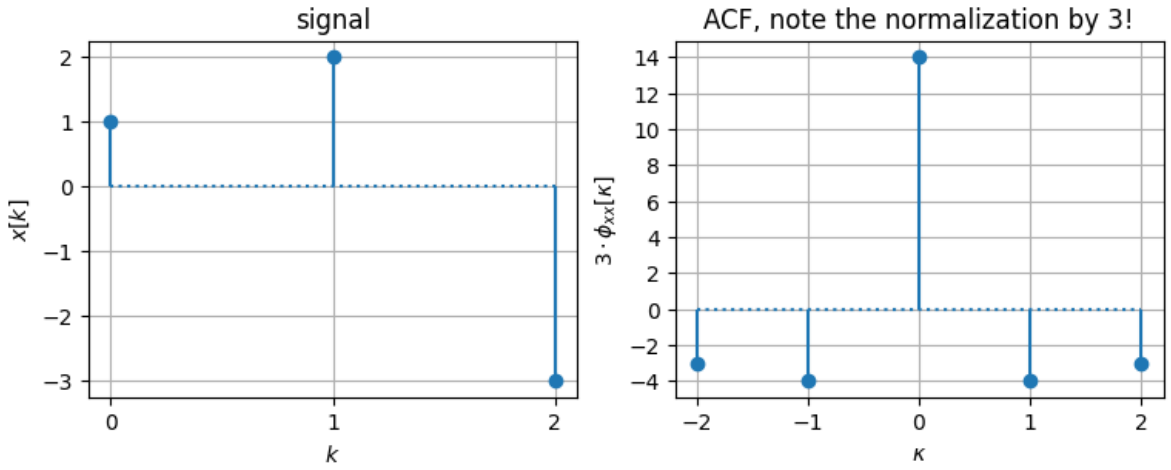
Picture 15. The result

4.4. ACF of a Short Sequence

```
x = (+1., +2., -3.)
kappa, acf = my_xcorr2(x, x, 'biased')  # use own function defined above

for i in acf:
    print('{:f}'.format(i), end=', ')

plt.figure(figsize=(9, 3))
plt.subplot(1, 2, 1)
plt.stem(x, basefmt='C0:')
plt.xticks(np.arange(0, 3))
plt.yticks(np.arange(-3, 3, 1))
plt.xlabel(r'$k$')
plt.ylabel(r'$x[k]$')
plt.title('signal')
plt.grid(True)

plt.subplot(1, 2, 2)
# note that we plot 3*ACF ! this gives simple integer results in the plot
plt.stem(kappa, acf*3, basefmt='C0:')
plt.xticks(np.arange(-2, 3))
plt.yticks(np.arange(-4, 16, 2))
plt.xlabel(r'$\kappa$')
plt.ylabel(r'$3 \cdot \phi_{xx}[\kappa]$')
plt.title('ACF, note the normalization by 3!')
plt.grid(True)
```

Picture 16. Implemented code

```
-1.000000, -1.333333, 4.666667, -1.333333, -1.000000,
```

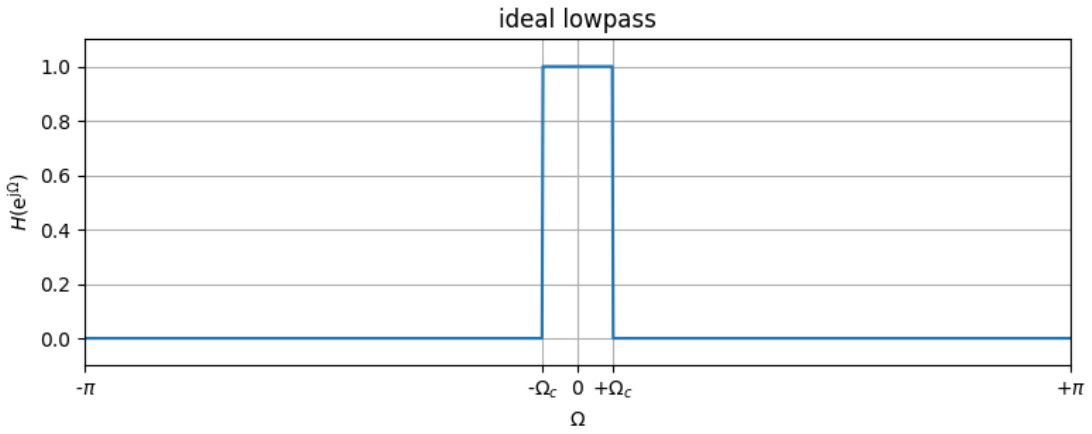


Picture 17. The result

4.5. Auto Correlation Function of LTI System's Output

```
Omegac = np.pi/14
N = 2**10
Omega = np.arange(N) * 2*np.pi/N - np.pi  # [-pi...pi)
H = np.ones(N)
H[Omegac < np.abs(Omega)] = 0
plt.figure(figsize=(9, 3))
plt.plot(Omega, H)
plt.xlabel(r'$\Omega$')
plt.ylabel(r'$H(\mathrm{e}^{\mathrm{j}\Omega})$')
plt.title('ideal lowpass')
plt.xticks([-np.pi, -Omegac, 0, +Omegac, +np.pi],
           [r'-$\pi$', r'-$\Omega_c$', '0', r'+$\Omega_c$', r'+$\pi$'])
plt.xlim(-np.pi, +np.pi)
plt.ylim(-0.1, 1.1)
plt.grid(True)
```

Picture 18. Implemented code

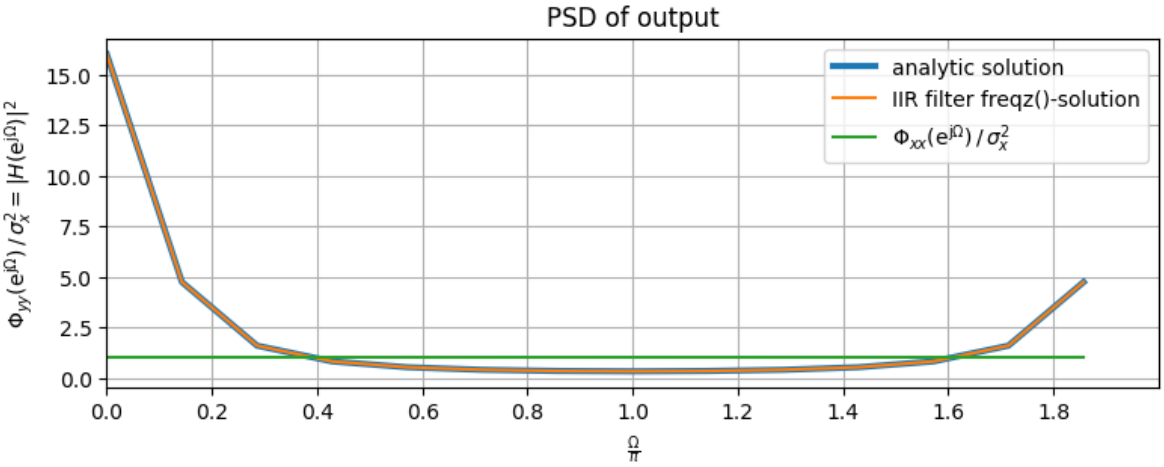


Picture 19. The result

4.6. Transfer Function of LTI System, Output PSD

```
N = 14
Omega = np.arange(N) * 2*np.pi/N
H2 = 2 / (25/8 - 3*np.cos(Omega))  # analytic
Omega, H_IIR = signal.freqz(b=(1), a=(1, -3/4), worN=Omega)  # numeric

plt.figure(figsize=(9, 3))
plt.plot(Omega/np.pi, H2, lw=3, label='analytic solution')
plt.plot(Omega/np.pi, np.abs(H_IIR)**2, label='IIR filter freqz()-solution')
plt.plot(Omega/np.pi, Omega*0+1,
         label=r'$\Phi_{xx}(\mathrm{e}^{\mathrm{j}\Omega})\,/\,\sigma_x^2$')
plt.xlabel(r'$\frac{\Omega}{\pi}$')
plt.ylabel(
    r'$\Phi_{yy}(\mathrm{e}^{\mathrm{j}\Omega})\,/\,\sigma_x^2 = |H(\mathrm{e}^{\mathrm{j}\Omega})|^2$')
plt.title('PSD of output')
plt.xlim(0, 2)
plt.xticks(np.arange(0, 20, 2)/10)
plt.legend()
plt.grid(True)
```

Picture 20. Implemented code

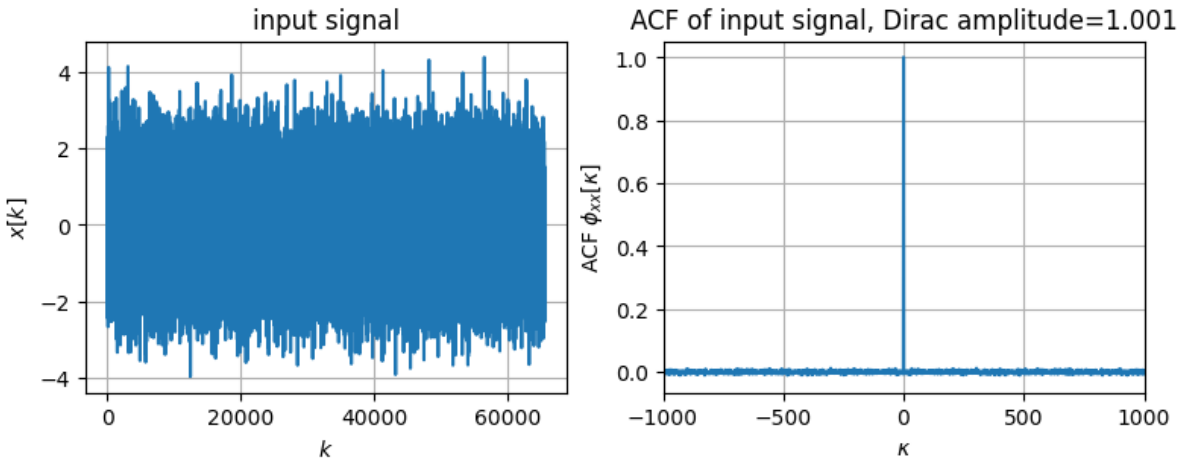


Picture 21. The result

4.7. Impulse Response Estimation with Random Signal in Time Domain

```
np.random.seed(2)  # arbitrary choice
Nx = 2**16
k = np.arange(Nx)
x = np.random.randn(Nx)
kappa, phixx = my_xcorr2(x, x, 'biased')  # we use biased here, i.e. 1/N normalization
idx = np.where(kappa==0)[0][0]

plt.figure(figsize=(9, 3))
plt.subplot(1, 2, 1)
plt.plot(k, x)
plt.xlabel('$k$')
plt.ylabel('$x[k]$')
plt.title('input signal')
plt.grid(True)
plt.subplot(1, 2, 2)
plt.plot(kappa, phixx)
plt.xlim(-1000, +1000)
plt.xlabel('$\kappa$')
plt.ylabel('ACF $\phi_{xx}[\kappa]$')
plt.title('ACF of input signal, Dirac amplitude=%4.3f' % phixx[idx])
plt.grid(True)
```

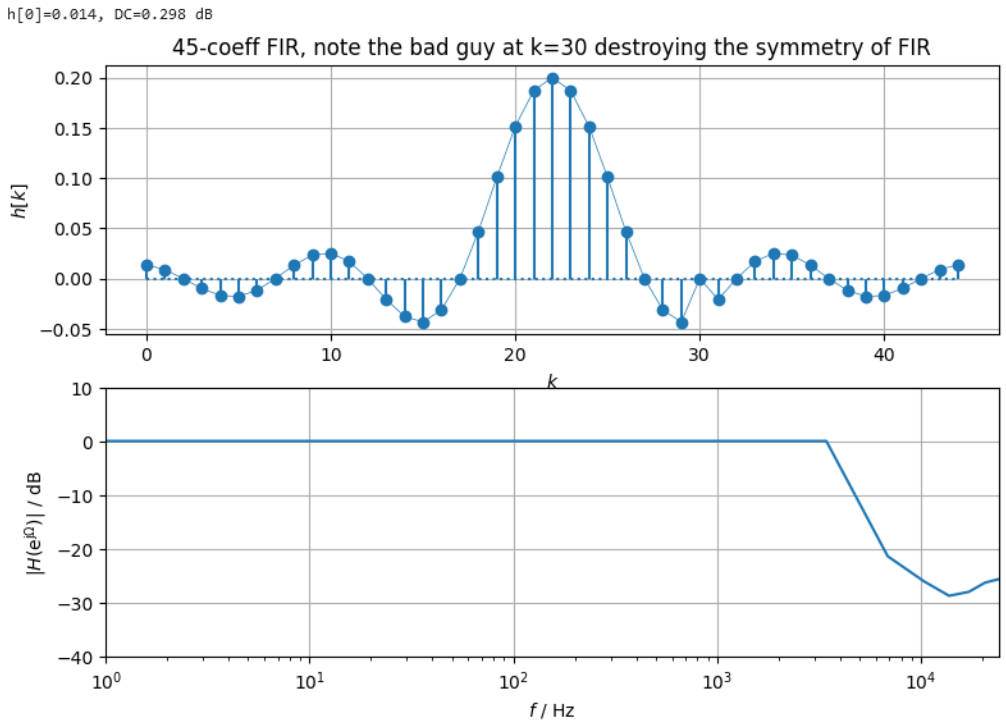
Picture 22. Implemented code



Picture 23. The result

```
fs = 48000  # sampling frequency in Hz
fc = 4800  # cut frequency in Hz
number_fir_coeff = 45  # FIR taps
h = signal.firls(numtaps=number_fir_coeff,  # example for demo
                 bands=(0, fc, fc+1, fs//2),
                 desired=(1, 1, 0, 0),
                 fs=fs)
Nh = h.size
k = np.arange(Nh)
# make the IR unsymmetric by arbitray choice for demonstration purpose
idx = 30
h[idx] = 0  # then FIR is not longer linear-phase, see the spike in the plot

print('h[0]={0:4.3f}, DC={1:4.3f} dB'.format(h[0], 20*np.log10(np.sum(h))))

N = 14
Omega = np.arange(0, N) * 2*np.pi/N
_, H = signal.freqz(b=h, a=1, worN=Omega)

plt.figure(figsize=(9, 6))
plt.subplot(2, 1, 1)
plt.stem(k, h, basefmt='C0:')
plt.plot(k, h, 'C0-', lw=0.5)
plt.xlabel(r'$k$')
plt.ylabel(r'$h[k]$')
plt.title(str(Nh)+'-coeff FIR, note the bad guy at k=%d destroying the symmetry of FIR' % idx)
plt.grid(True)
plt.subplot(2, 1, 2)
plt.semilogx(Omega / (2*np.pi) * fs, 20*np.log10(np.abs(H)))
plt.xlabel(r'$f$ / Hz')
plt.ylabel(r'$|H(\mathrm{e}^{\mathrm{j}\Omega})|$ / dB')
plt.xlim(1, fs//2)
plt.ylim(-40, 10)
plt.grid(True)
```

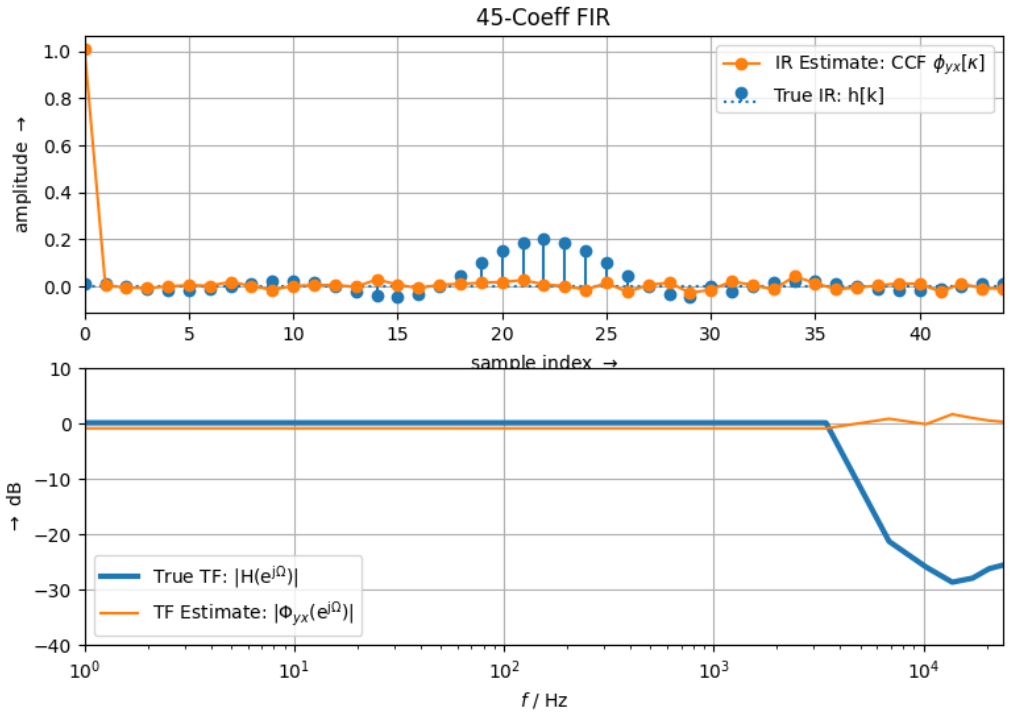
Picture 24. Implemented code



Picture 25. The result

```
kappa, phiyx = my_xcorr2(y, x, 'biased')  # get cross correlation in order y,x

# find the index for kappa=0, the IR starts here
idx = np.where(kappa == 0)[0][0]
# cut out the IR, since we know the numtaps this is easy to decide here
h_est = phiyx[idx:idx+Nh] / len(y)
# get DTFT estimate of PSD
_, Phiyx = signal.freqz(b=h_est, a=1, worN=Omega)

plt.figure(figsize=(9, 6))
plt.subplot(2, 1, 1)
plt.stem(h, basefmt='C0:', label='True IR: h[k]')
plt.plot(kappa, phiyx / len(y), 'C1o-',
         label=r'IR Estimate: CCF $\phi_{yx}[\kappa]$')
plt.xlim(0, Nh-1)
plt.xlabel(r'sample index $\rightarrow$')
plt.ylabel(r'amplitude $\rightarrow$')
plt.title(str(Nh)+'-Coeff FIR')
plt.legend()
plt.grid(True)
plt.subplot(2, 1, 2)
plt.semilogx(Omega/2/np.pi*fs, 20*np.log10(np.abs(H)), lw=3,
             label=r'True TF: $|\mathrm{H}(\mathrm{e}^{\mathrm{j}\Omega})|$')
plt.semilogx(Omega/2/np.pi*fs, 20*np.log10(np.abs(Phiyx)),
             label='TF Estimate: $|\Phi_{yx}(\mathrm{e}^{\mathrm{j}\Omega})|$')
plt.xlabel(r'$f$ / Hz')
plt.ylabel(r'$\rightarrow$ dB')
plt.xlim(1, fs//2)
plt.ylim(-40, 10)
plt.legend()
plt.grid(True)
```

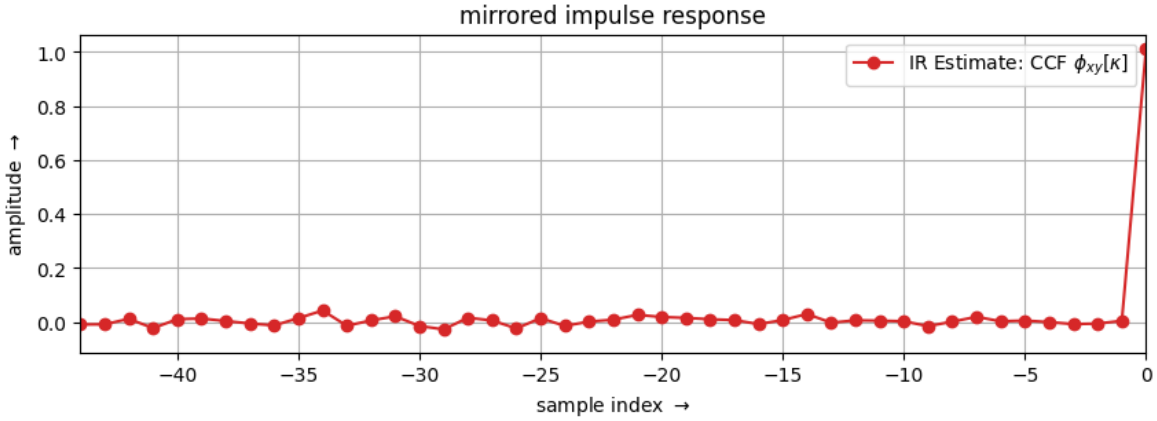
Picture 26. Implemented code



Picture 27. The result

```
kappa, phixy = my_xcorr2(x, y, 'biased')  # get cross correlation x,y

plt.figure(figsize=(10, 3))
plt.plot(kappa, phixy/len(y), 'C3o-',
         label=r'IR Estimate: CCF $\phi_{xy}[\kappa]$')
plt.xlim(-(Nh-1), 0)
plt.xlabel(r'sample index $\rightarrow$')
plt.ylabel(r'amplitude $\rightarrow$')
plt.title('mirrored impulse response')
plt.legend()
plt.grid(True)
```

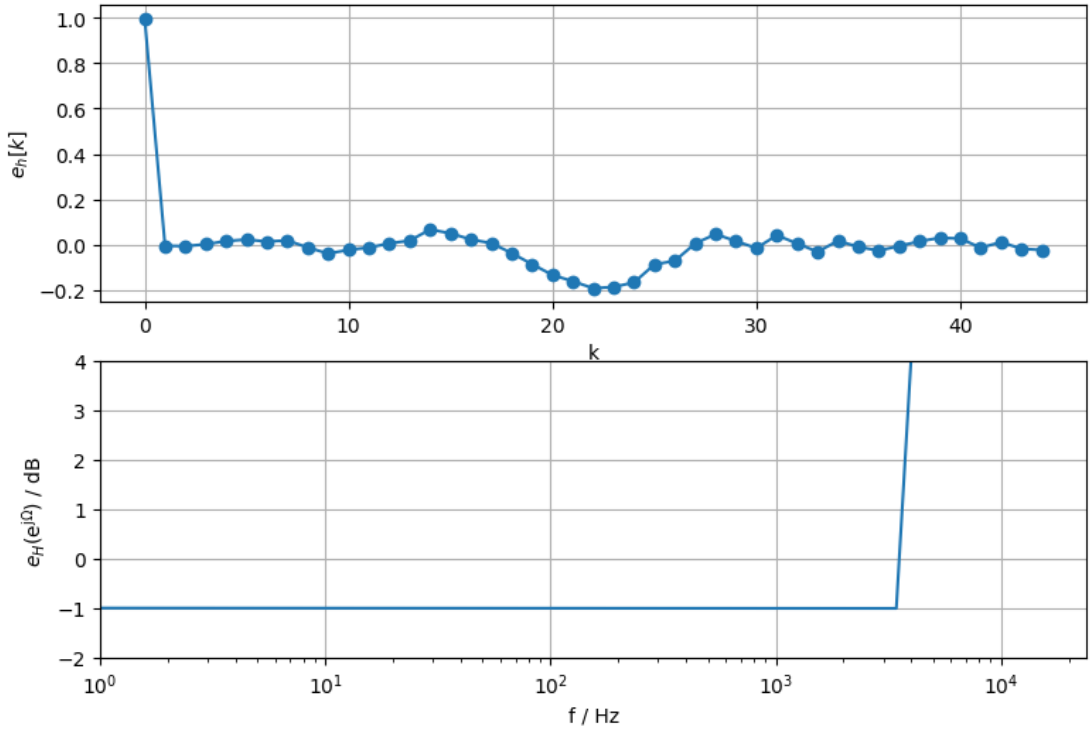
Picture 28. Implemented code



Picture 29. The result

```
plt.figure(figsize=(9, 6))
plt.subplot(2, 1, 1)
plt.plot(h_est - h, 'o-')
plt.xlabel('k')
plt.ylabel(r'$e_h[k]$')
plt.grid(True)
plt.subplot(2, 1, 2)
plt.semilogx(Omega / (2*np.pi) * fs, 20 *
             np.log10(np.abs(Phiyx)) - 20*np.log10(np.abs(H)))
plt.xlabel('f / Hz')
plt.ylabel(r'$e_H(\mathrm{e}^{\mathrm{j}\Omega})$ / dB')
plt.xlim(1, fs//2)
plt.ylim(-2, 4)
plt.grid(True)
```

Picture 30. Implemented code



Picture 31. The result

5. Conclusions

In this lab, we delved into the behavior of random signals when subjected to Linear Time-Invariant (LTI) systems, utilizing various numerical characteristics to analyze their responses comprehensively. Through this exploration, we aimed to gain insights into how LTI systems interact with random signals, shedding light on their dynamic behavior and the effects of system parameters on signal processing outcomes.