lab4

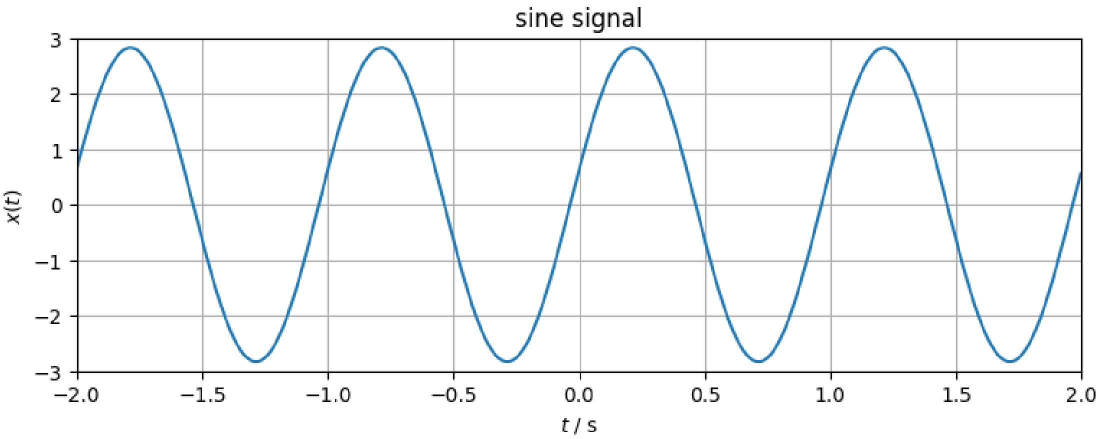
April 14, 2024

[1]:
```
# most common used packages for DSP, have a look into other scipy submodules
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import 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
```

[2]:
```
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))
```

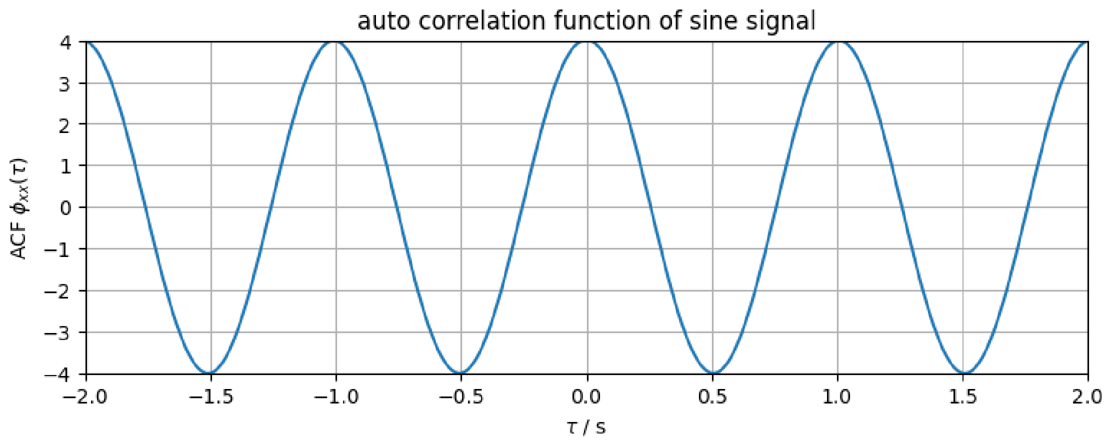
```python
plt.plot(t, x)
plt.xlim(-2, 2)
plt.ylim(-3, 3)
plt.xlabel(r'$t$ / s')
plt.ylabel(r'$x(t)$')
plt.title('sine signal')
plt.grid(True)
```

```
fs = 256.000000 Hz
```



[3]:

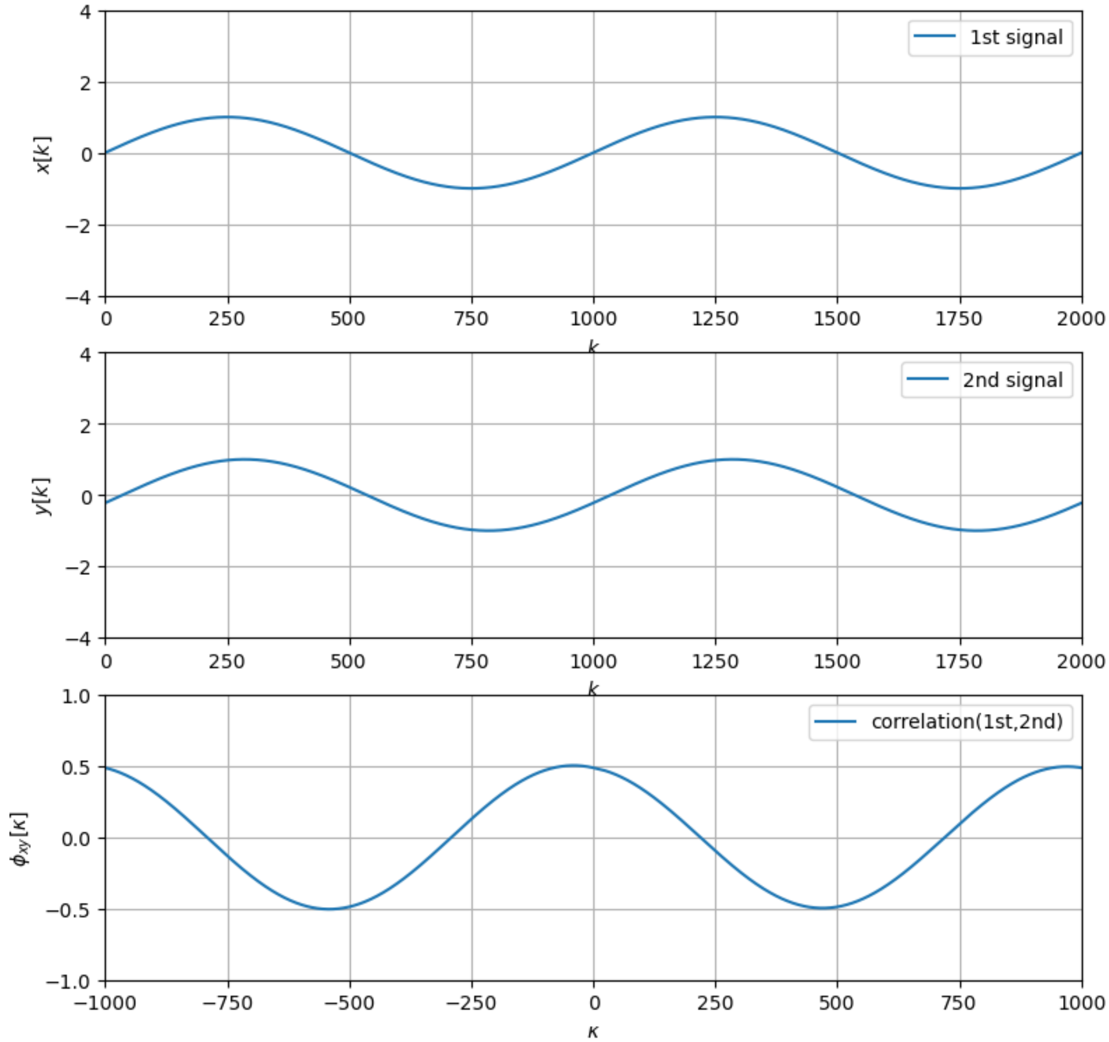
```python
kappa, phixx = my_xcorr2(x, x, 'unbiased')

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)
```



[4]:
```python
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)
```

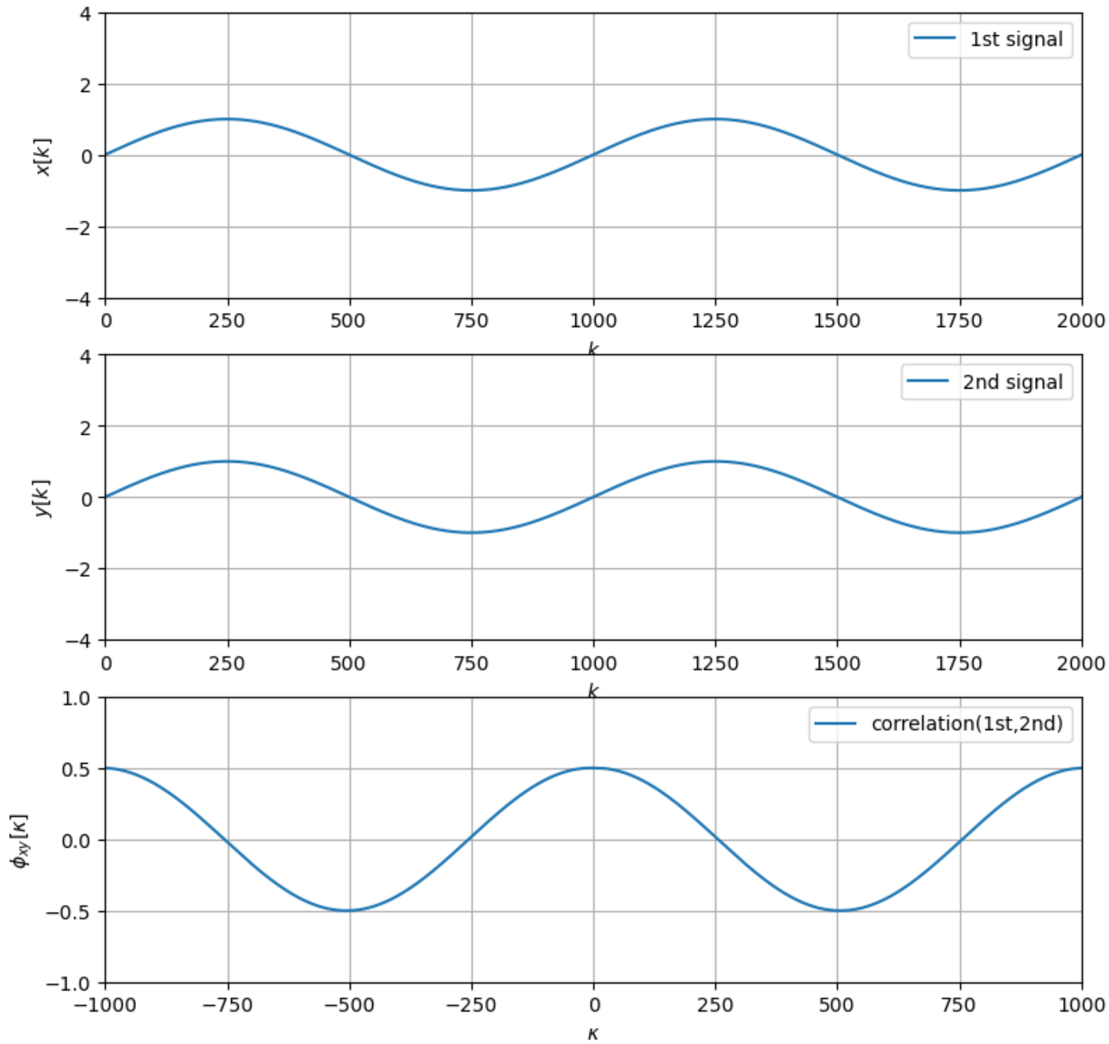
[5]:
```python
N = 5000
k = np.arange(N)
Omega1 = 5 * 2*np.pi/N
Omega2 = 20 * 2*np.pi/N
```

[6]:
```python
# 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')
```

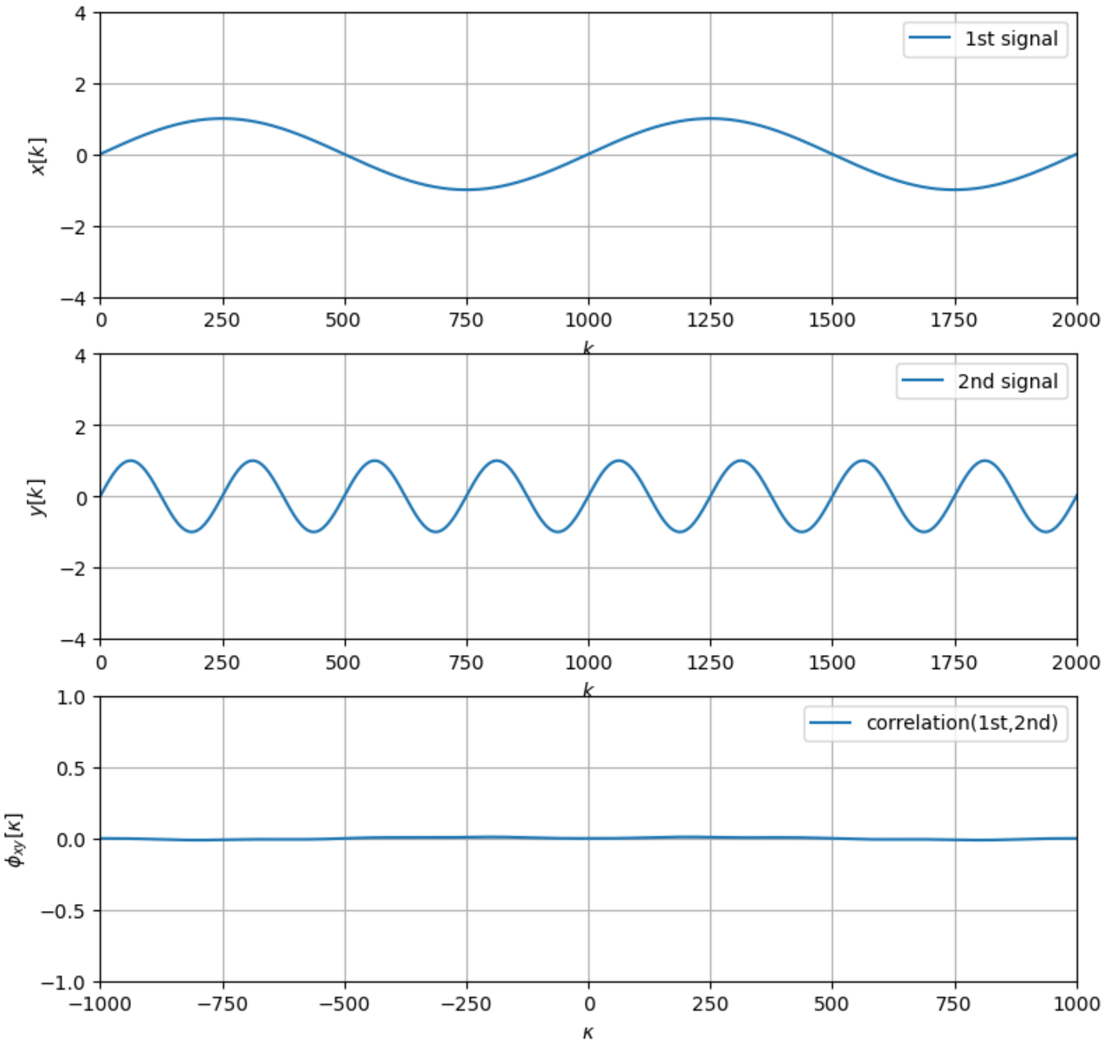


[7]:
```python
# f)
x = np.sin(Omega1*k)
```

```
y = x
my_ccf_plot(x, y, scaleopt='unbiased')
```

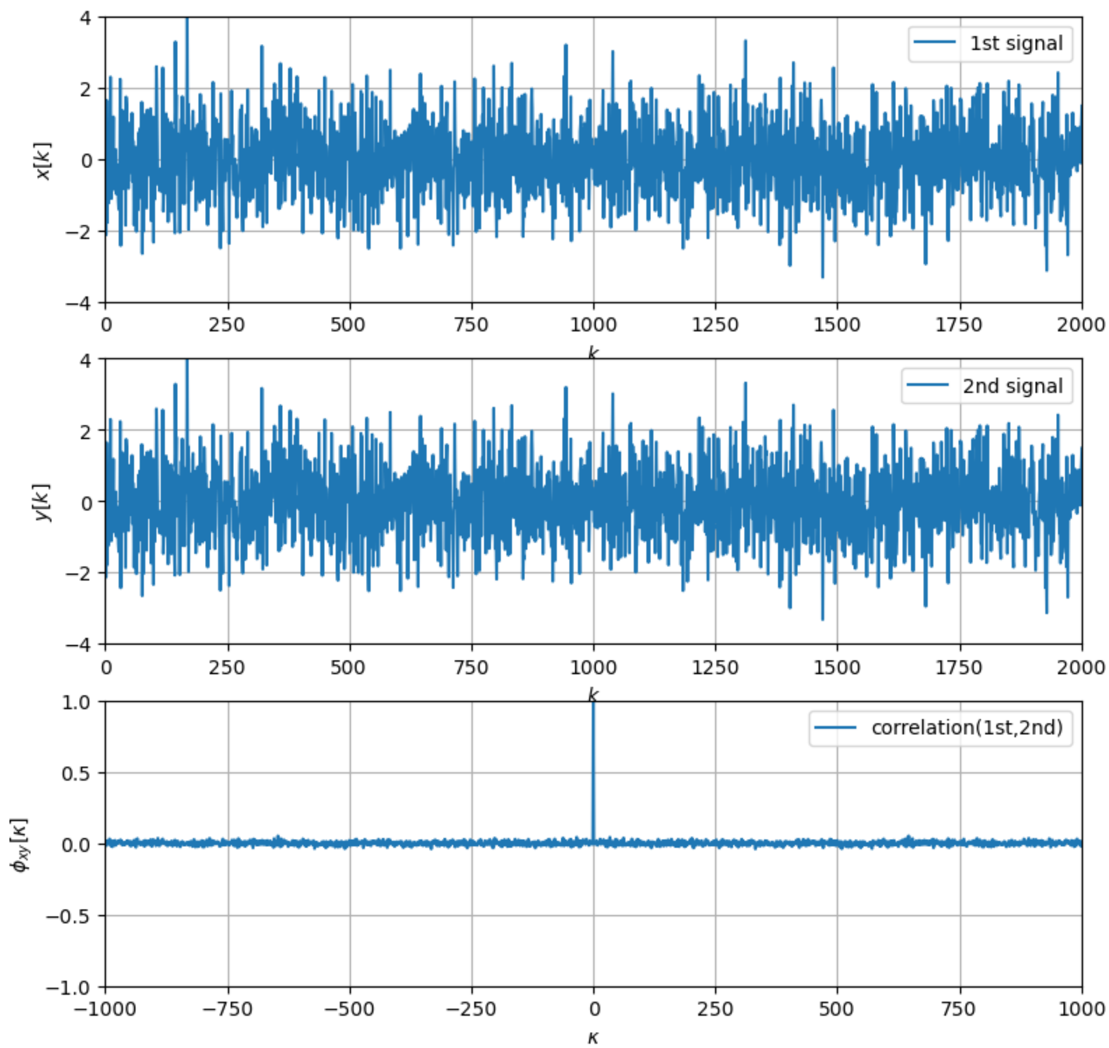


[8]:
```
# g)
x = np.sin(Omega1*k)
y = np.sin(Omega2*k)
my_ccf_plot(x, y, scaleopt='unbiased')
```



[9]:

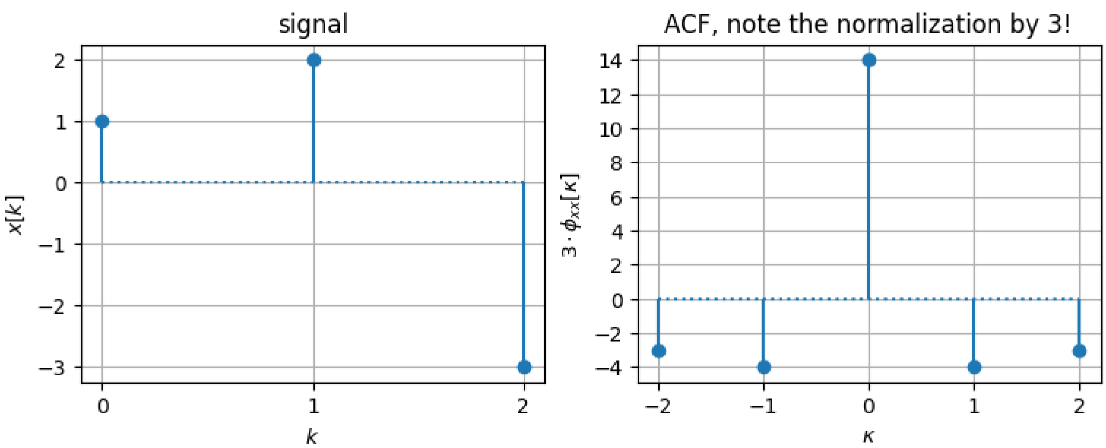
```
# h)
np.random.seed(2)  # arbitrary choice
x = np.random.randn(N)
y = x
my_ccf_plot(x, y, scaleopt='biased')
```



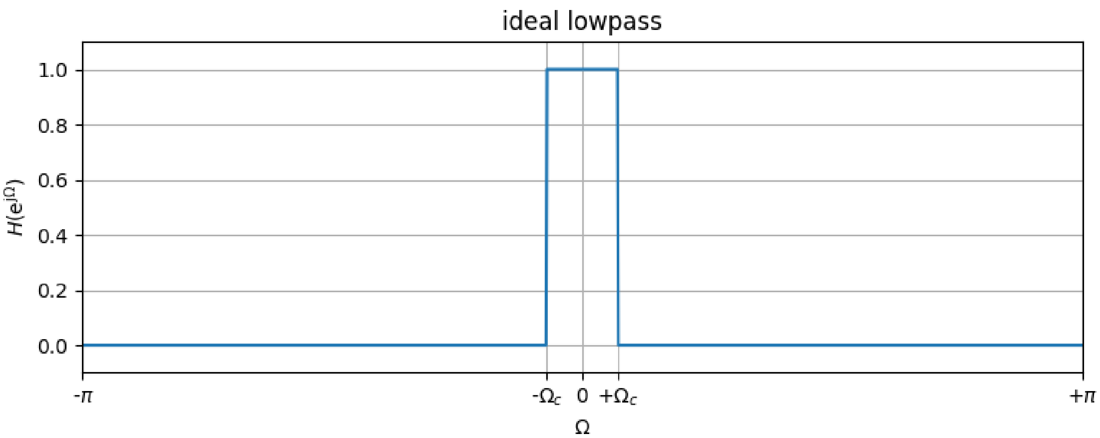
[10]:
```python
x = (+1., +2., -3.)
kappa, acf = my_xcorr2(x, x, 'biased')  # use own function defined above

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)
```

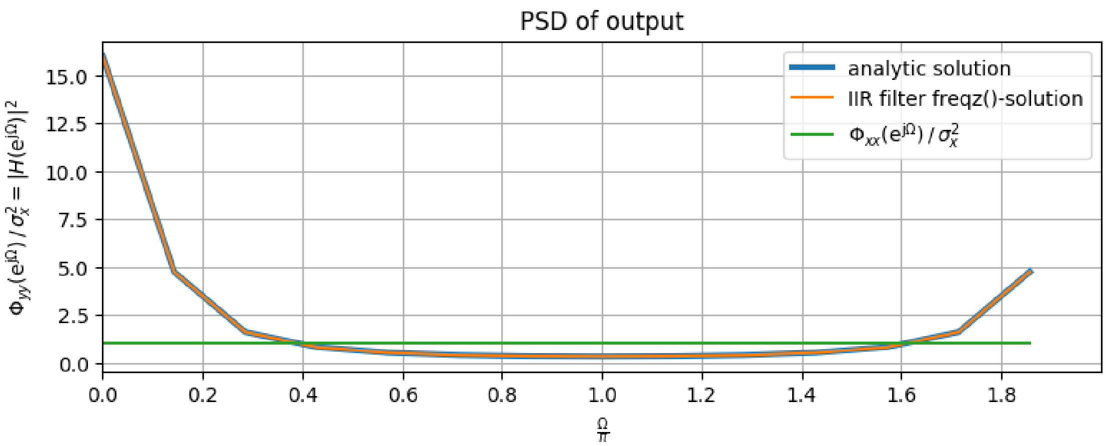
-1.000000, -1.333333, 4.666667, -1.333333, -1.000000,



[11]:
```
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)
```

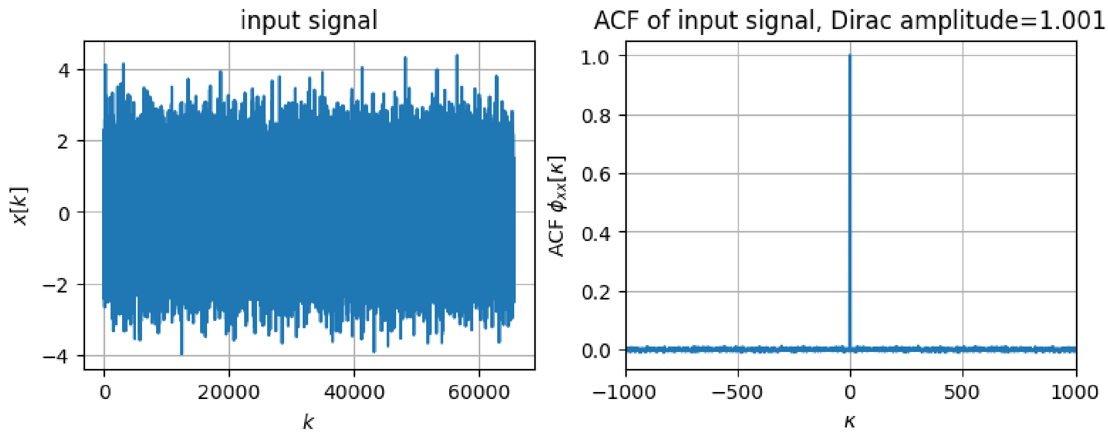


[12]:
```
N = 14
Omega = np.arange(N) * 2*np.pi/N
H2 = 2 / (25/8 - 3*np.cos(Omega))  # analytic
Omega, H_IIR = signal.freqz(b=(1), a=(1, -3/4), worN=Omega)  # numeric

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)
```



[13]:

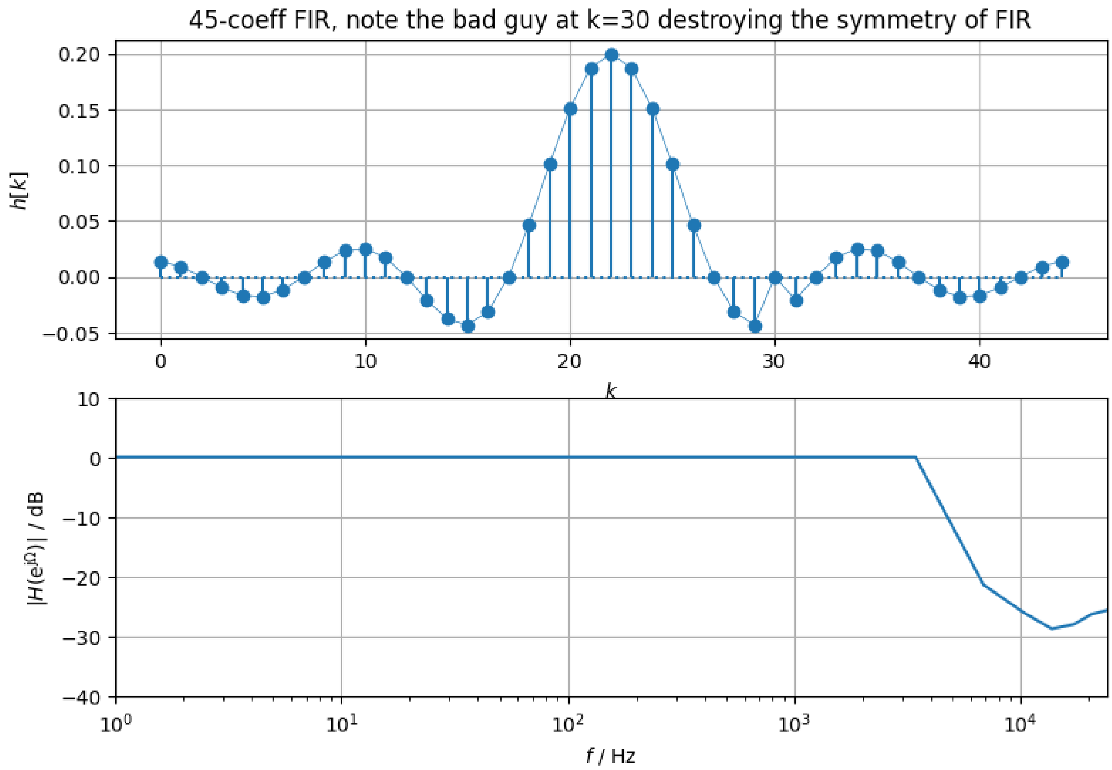
```
np.random.seed(2)  # arbitrary choice
Nx = 2**16
k = np.arange(Nx)
x = np.random.randn(Nx)
kappa, phixx = my_xcorr2(x, x, 'biased')  # we use biased here, i.e. 1/N␣
↪normalization
idx = np.where(kappa==0)[0][0]

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)
```



[14]:
```
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')
```

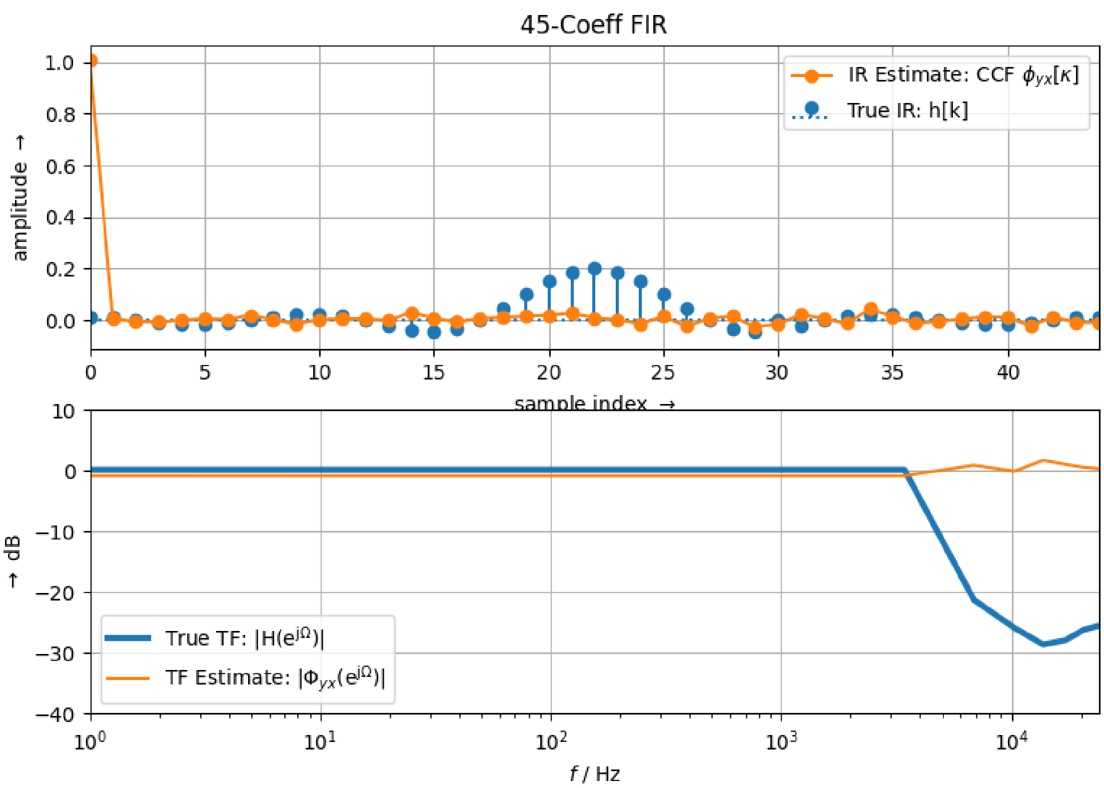
```
plt.ylabel(r'$|H(\mathrm{e}^{\mathrm{j}\Omega})|$ / dB')
plt.xlim(1, fs//2)
plt.ylim(-40, 10)
plt.grid(True)
```

h[0]=0.014, DC=0.298 dB



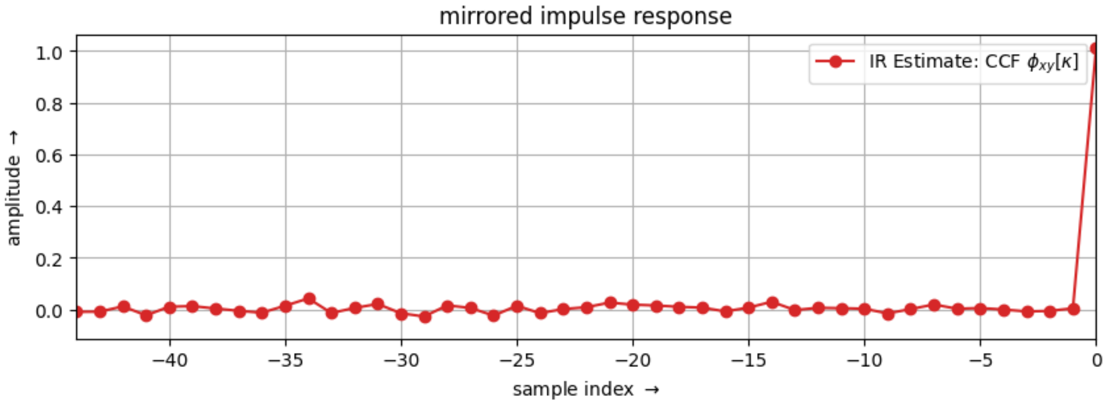
[15]:
```
kappa, phiyx = my_xcorr2(y, x, 'biased')  # get cross correlation in order y,x

# 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)
```

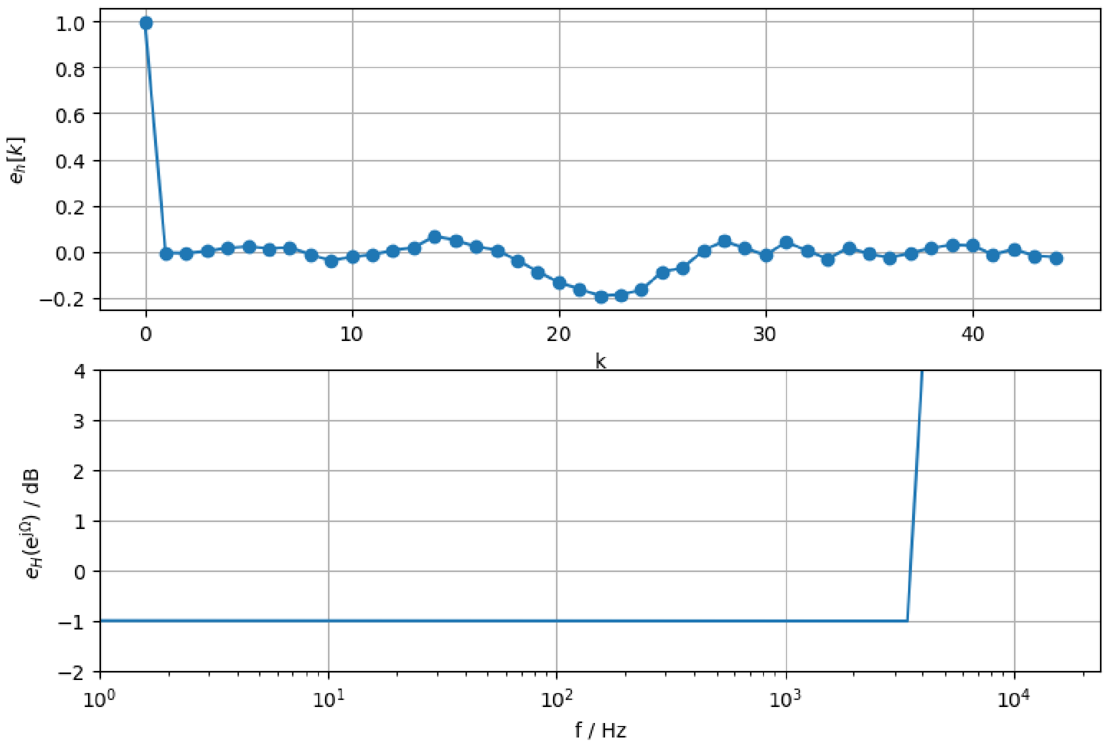


[16]:
```
kappa, phixy = my_xcorr2(x, y, 'biased')  # get cross correlation x,y

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)
```



[17]:
```
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)
```



1.0
0.8
0.6
0.4
0.2
0.0
−0.2
$e_h[k]$
0
10
20
30
40
k
4
3
2
1
0
−1
−2
$e_H(e^{j\Omega})$ / dB
10^0
10^1
10^2
10^3
10^4
f / Hz

[ ]: