SC3011TN_PythonAssignment2_Denoising

March 4, 2021

1 SC3011TN - Stochastische Signaalanalyse - 2020/2021

2 Group details

Student 1:

```
Name: Jeroen Sangers

Student number: 4645197
```

Student 2:

```
Name: Reinaart van Loon

Student number: 4914058
```

Date of completion: March 4th

2.1 Introduction

In this exercise, we consider an RF communication system consisting of an emitter and a receiver. Interference originating from the RF environment and from thermal noise of the system cause signal degradation at the receiver. Since the nature of these aberrations is stochastic, Wiener filtering can be used to subtract the effects of the distortion.

In the configuration of the figure below, all signals are discrete-time and real. The corrupted signal $x(n) = d(n) + v(n)$ contains both the signal of interest $d(n)$ and noise $v(n)$, which is uncorrelated to d(n). In this assignment you will design a FIR filter $W(z)$, used to estimate the ground truth signal from $x(n)$ and from the statistical properties of the noise.

2.2 First steps

Make sure that the files `Audio.py`, `Communication.py` and `denoising-scheme.jpg`, as well as the recorded signals `gong_5cm.mat`, `gong_100cm.mat`, `gong_250cm.mat` and `noise.mat` are included in the same directory as this Jupyter notebook. Run the following code:

[ ]:
```
! pip install pyserial
! pip install sounddevice
! pip install scipy
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Note: You only need to execute this code snippet once, after that your python will permanently include the modules. If the the code gives an error, manually install the modules using pip in the command prompt. In Windows: start button->search, type cmd->`pip install package_name`

3 Obtaining the data

In this section, you will import data from the transmission experiment, described in the introduction. As the COVID-measures prevent us from letting you record your own signals, we have done the experiment for you and we provide you with the raw recordings. Before doing so, we describe the experimental setup that we used to record the signals. The experimental setup is shown in the figure below.

For this experiment, two Arduino MK1000's are used. These Arduino microcontrollers are able to send and receive signals through radio frequency (RF) wireless transmission. As such, the first microcontroller is configured as a sender, used to send a wav-file `gong.wav`. The second microcontroller is configured as a receiver and it is linked to the sender. By simultaneously activating both microcontrollers, the receiver will record the signal sent by the sender.

However, the signal transmission is affected by RF noise. Examples of sources of this noise are disturbances in the RF band, distance-dependent noise caused by the distance between the microcontrollers and thermal distortion of the signal. These noise sources act additively on the sent signal `gong.wav`. For distances of 5 cm, 100 cm and 250 cm between the microcontrollers, we recorded `gong_5cm.mat`, `gong_100cm.mat` and `gong_250cm.mat` respectively, which you can listen to later in this assignment.

To filter the noise from the signal using denoising, we also need a recording of the noise for every distance of 5 cm, 100 cm and 250 cm between the microcontrollers. The total noise has a complicated spectrum, so to save you some mathematical work we solely focus on ambient, distance-independent noise. This noise is assumed to be zero-mean and white, and a measurement of this noise is contained in `noise.mat`.

NB. The recorded values in `gong_5cm.mat`, `gong_100cm.mat`, `gong_250cm.mat` and `noise.mat` are expressed in decibels relative to full scale (dB-FS), which is a unit used to expressed signal amplitudes in digital systems.

With these signals, you have enough information to calculate the weights of a FIR Wiener filter. This assignment will guide you through this process.

3.0.1 Play the signal

During the assignment, you will want to listen to the raw and filtered signals. The function `play_signal` can be used for this. Run the cell below to define the function.

[3]:
```
#import sounddevice as sim
import sounddevice as sd

# function for playing a signal
```

```python
# You can use it further if you want to listen smth

def play_signal( signal ):
    sd.play( signal.astype('int16'), 22050 )
```

3.0.2 Loading files

Run this section in order to load the .mat files with the noise audios and the background noise signals.

[4]:
```python
# packages
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio

# modeling distance in centimeters (cm)
distances = np.array( [5.0, 100.0, 250.0] )

# number of experiments with different distances
N_exp = distances.shape[0]

print( f'Number of experiments = {N_exp}' )

# list of files for all distance experiments with audio
filenames = ['gong_5cm.mat', 'gong_100cm.mat', 'gong_200cm.mat']

# Number of data points
N = len( sio.loadmat( filenames[0] )['audio'][0] )

# create array with received audio signals
data_noised = np.zeros( (N_exp, N), dtype=float )
for iexp in range(N_exp):
    data_noised[iexp] = sio.loadmat( filenames[iexp] )['audio'][0]

# array for the time steps
time_steps = np.linspace( 0, N-1, N ).astype( int )
print( f'\nN = {N}' )
```

```
Number of experiments = 3

N = 65411
```

[5]:
```python
# load background noise signal that was measured

filename_noise = 'noise.mat'
data_mat = sio.loadmat( filename_noise )['audio'][0]
```

```
Nnoise = len( data_mat )
noise_measured = np.array( data_mat, dtype=float )

print( f'Number of noise time steps = {Nnoise}' )
```

```
Number of noise time steps = 88200
```

4 Exercises

This part should be handled after all audio files were uploaded into *data_noised* array and the noise was uploaded into *noise_measured* file.

4.1 Question 1

Formulate the denoising problem as a Minimum Variance FIR Wiener Problem, using the symbols $x(n)$, $v(n)$, $d(n)$, $\hat{d}(n)$, $e(n)$ as in the figure presented in the **Introduction**. In particular * Explain for each variable what signal they represent * Write the cost function $J(W(z))$ and the optimization problem that should be solved to find the Minimum Variance FIR Wiener Filter.

Note that no numerical values are required, but you need to specify the meaning of all the symbols and their relation to the sent and received data. **(2 points)**

4.1.1 Answer 1

- $d(n)$ = signal of interest
- $v(n)$ = ZMWN that corrupts received signal
- $x(n)$ = the signal received by the receiver
- $\hat{d}(n)$ = the received signal flitered by the Wiener filter with transfer function $W(z)$
- $e(n)$ = the error, it denotes the difference between the actual signal and the received signal filtered by $W(z)$

Cost function: $J(W(z)) = E[|d(n) - \hat{d}(n)|^2] = E[|e(n)|^2]$

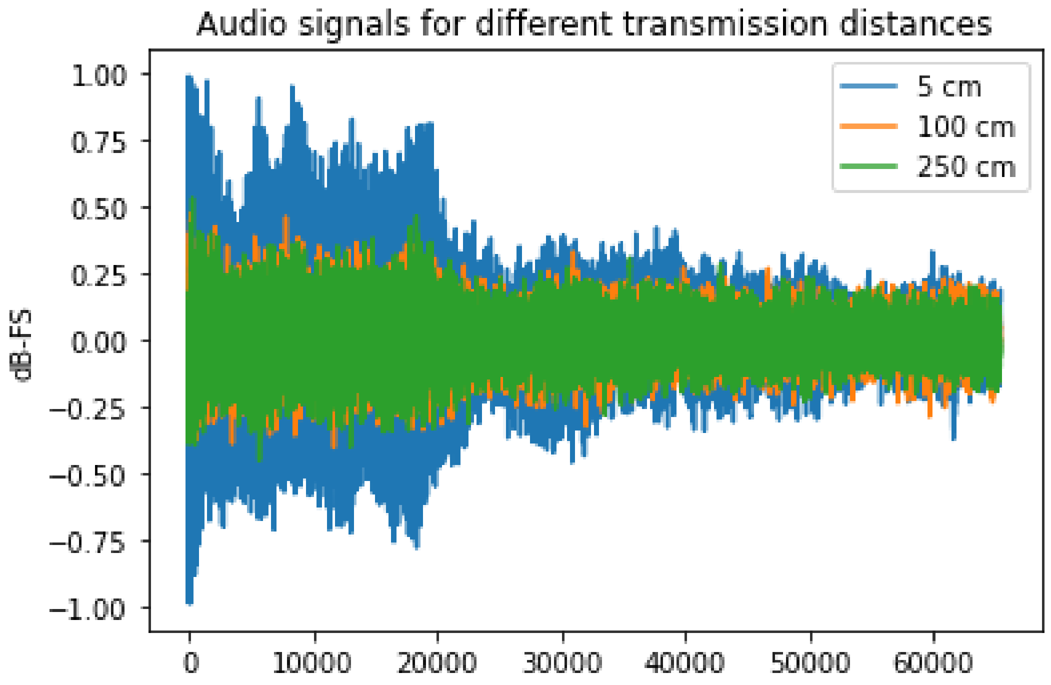
4.2 Question 2

Plot the received audio signals and the received noise signal. You can plot all the audio signals in the same figure, but use a separate plot for the noise signal. Do not forget to label the axes and to provide a legend. **(1 point)**

NB. The recorded values in `gong_5cm.mat`, `gong_100cm.mat`, `gong_250cm.mat` and `noise.mat` are expressed in decibels relative to full scale (dB-FS).

4.2.1 Answer 2

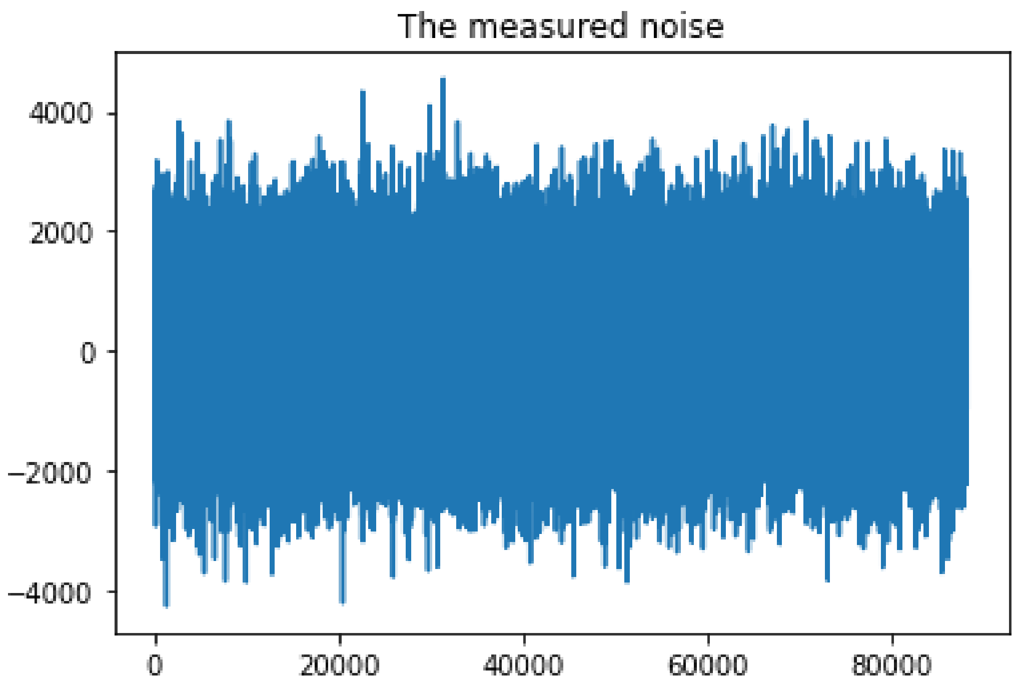
[6]:
```
# plot noised audio signal

plt.plot(data_noised[0]*2**(-15), label= '5 cm')
plt.plot(data_noised[1]*2**(-15), label='100 cm')
plt.plot(data_noised[2]*2**(-15), label='250 cm')
plt.legend()
plt.title('Audio signals for different transmission distances')
plt.ylabel('dB-FS')
plt.show()
```



[7]:
```
# listen the noised audio signals
# Warning to headphone users: depending on your own sound configurations, the␣
 ↪played signal may be very loud!
# Turn down your volume the first time you try playing a signal.

#ind_to_listen = 2 #Choose 0 (5 cm), 1 (100 cm) or 2 (250 cm).
#play_signal( data_noised[2] )
```

[8]:
```
# plot the measured noise signal
plt.title('The measured noise')
plt.plot(noise_measured)
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f11f14f3730>]
```



4.3 Question 3

Determine the Wiener-Hopf equation for the denoising case with filter order m. Clearly specify the size and content of all quantities involved.

What assumptions are we making for the (true and noised) audio signal and the noise in order to obtain the weights of the filter w?

Derive how can we calculate the right hand side of the Wiener-Hopf equation, assuming that we know $r_x(k)$, $\forall k$ and the noise variance σ_v^2. **(2 points)**

4.3.1 Answer 3

a) $R_x \hat{w} = r_{dx}$

With:

1. $R_x = E[x^*(n)x(n)^T]$: covariance matrix of vector $x(n)$, it has a size of $m \times m$
2. $x(n)$: The vector with the received datapoints from the present $(t = n)$ to m values back in the past$(t = n - m + 1)$
3. $\hat{w}$: Vector containing the parameters of the Wiener filter (length m)
4. $r_{dx} = E[d(n)x^*(n)]$: The cross correlation with a zero lag of the signal of interest $d(n)$ and the received datapoints $x(n)$ from the present $(t = n)$ to m values back in the past $(t = n-m+1)$

b) We assume that $v(n)$ and $d(n)$ are uncorrelated.

c)

$$
\begin{aligned}
r_{dx}(k) &= E[d(n)\big(v(n-k)+d(n-k)\big)] \\
&= E[d(n)\cdot v(n-k) + d(k)\cdot d(n-k))] \\
&= r_{dv} + r_d \\
&= r_d \quad (r_{dv} = 0\ , because\ d(n)\ and\ v(n)\ uncorrelated)
\end{aligned}
$$

We also have:

$$
\begin{aligned}
r_x(k) &= E[\big(v(n)+d(n)\big)\big(v(n-k)+d(n-k)\big)] \\
&= E[d(n)\cdot v(n-k) + d(k)\cdot d(n-k) + v(n)\cdot d(n-k) + v(n)\cdot v(n-k)] \\
&= r_{dv} + r_d + r_{vd} + r_v \\
&= r_d + r_v \quad (r_{dv} = r_{vd} = 0\ , because\ d(n)\ and\ v(n)\ uncorrelated)
\end{aligned}
$$

Therefore we have: $r_d = r_x - r_v$

Since we assume $v(n)$ to be ZMWN we use $r_v(k) = \sigma_v^2\,\delta(k)$ to find:

$r_d = r_x - \sigma_v^2\,\delta(k)$

4.4 Question 4

Estimate the mean and variance (σ_v^2) of the measured noise. Print these estimates. **(1 point)**

4.4.1 Answer 4

$\hat{m}_v = -0.713$

$\hat{\sigma}_v^2 = 9.94 \cdot 10^5$

[9]:
```python
# estimate the mean and variance of the measured noise
from IPython.display import display, Math

noise_mean = np.average(noise_measured)
noise_var = np.average((noise_measured-noise_mean)**2)

display(Math((r'\hat{m}_v = %.3e,  \; \; \hat{\sigma}^2_v = %.3e' %(noise_mean,␣
↪noise_var))))
```

$\hat{m}_v = -7.130e-01,\ \ \hat{\sigma}_v^2 = 9.938e+05$

4.5 Question 5

1. Assuming the audio signals are ergodic, calculate the auto-correlation function $r_x(k)$ of every received audio signal for lags $0,\dots,9$. Print these values. **(2 points)**
2. Calculate the cross-correlation function $r_{dx}(k)$ of every received audio signal for lags $0,\dots,9$. Print these values. **(1 point)**
3. Using the formula below, compute and plot the auto-correlation of the noise $v(n)$ for lags $0,\dots,9$. Is the noise ZMWN? Explain your answer. **(1 point)**

Note: if the signal $x(n)$ is auto-correlation ergodic, the auto-correlation function $r_x(k)$ can be estimated as (if we have only values $x_1, x_2, \ldots, x_N$):

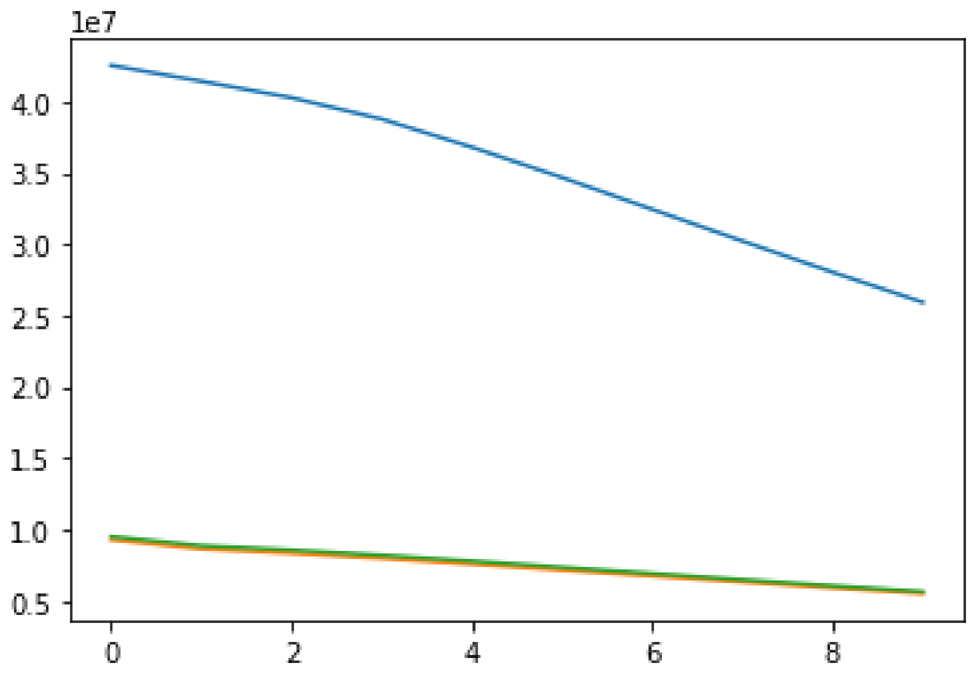
$$r_x(k) = \frac{1}{N-k} \sum_{i=k+1}^{N} x(i)x^*(i-k)$$

4.5.1 Answer 5

Click here to type your answer. Type *Markdown* and LaTeX: α^2.

[10]:
```
# calculating the auto-correlation function for the audio signals
from tabulate import tabulate

X = data_noised
r_x = np.zeros((3,10))

for i in range(10):
    k = i
    N = len(data_noised[0])
    r_x[:,i] = 1/(N-k) *np.sum(X[:,k:]*X[:,:(N-k)], axis=1)

plt.plot(r_x.T)
table = np.zeros((10,4))
table[:,1:] = r_x.T
table[:,0] = np.arange(10)
Headers = ["k","50cm","100cm","200cm"]
print("Table for r_x(k) \n")
print(tabulate(table, headers=Headers))
```

```
Table for r_x(k)

  k         50cm        100cm        200cm
---  -----------  -----------  -----------
  0  4.25393e+07  9.2997e+06   9.52128e+06
  1  4.14259e+07  8.65364e+06  8.87586e+06
  2  4.02718e+07  8.35578e+06  8.57613e+06
  3  3.87734e+07  7.99334e+06  8.20458e+06
  4  3.68007e+07  7.59432e+06  7.80073e+06
  5  3.46727e+07  7.17164e+06  7.36523e+06
  6  3.24518e+07  6.75583e+06  6.92713e+06
  7  3.02276e+07  6.3367e+06   6.48961e+06
  8  2.8042e+07   5.92169e+06  6.06339e+06
  9  2.59319e+07  5.53415e+06  5.6539e+06
```



```
[11]: # calculating the auto-correlation function for the noise
      X = noise_measured
      r_v = np.zeros(10)

      for i in range(10):
          k = i
          N = len(noise_measured)
          r_v[i] = 1/(N-k) *np.sum(X[k:]*X[:(N-k)])

      table = np.zeros((10,2))
      table[:,1] = r_v.T
      table[:,0] = np.arange(10)
      Headers = ["k","r_v(k)"]
      print("Table for r_v(k) \n")
      print(tabulate(table, headers=Headers))
```

```
Table for r_v(k)

  k      r_v(k)
---  ----------
  0  993765
  1   -1494.5
  2    4070.83
  3   -2920
  4   -4399.76
  5     158.426
  6     625.851
  7   -3770.77
  8    1334.58
  9    1170.53
```

[12]:
```
# calculating the cross-correlation function r_dx
r_dx = np.zeros((3,10))

for i in range(10):
    r_dx[:,i] = r_x[:,i]- r_v[i]

table = np.zeros((10,4))
table[:,1:] = r_dx.T
table[:,0] = np.arange(10)
Headers = ["k","50cm","100cm","200cm"]
print("Table for r_dx(k) \n")
print(tabulate(table, headers=Headers))
```

```
Table for r_dx(k)

  k         50cm        100cm        200cm
---  -----------  -----------  -----------
  0  4.15455e+07  8.30594e+06  8.52752e+06
  1  4.14274e+07  8.65513e+06  8.87735e+06
  2  4.02677e+07  8.3517e+06   8.57206e+06
  3  3.87763e+07  7.99626e+06  8.2075e+06
  4  3.68051e+07  7.59872e+06  7.80513e+06
  5  3.46726e+07  7.17148e+06  7.36507e+06
  6  3.24512e+07  6.7552e+06   6.9265e+06
  7  3.02314e+07  6.34047e+06  6.49338e+06
  8  2.80406e+07  5.92035e+06  6.06205e+06
  9  2.59307e+07  5.53298e+06  5.65273e+06
```

4.6 Question 6

Using a 10-th order FIR Wiener filter, calculate the optimal filter coefficients for every received audio signal. Print these values. **(2 points)**

Hint: The `scipy.linalg.toeplitz()` function may be useful.

4.6.1 Answer 6

Click here to type your answer. Type *Markdown* and LaTeX: α^2.

[13]:
```
# obtaining the optimal Wiener filter coefficients
import scipy.linalg as lin

#Assuming we rewrite Wiener-Hopf so solve for the \hat{w} vector
```

```
r_x_50 = r_x[0]
r_dx_50 = r_dx[0]

Rx = lin.toeplitz(r_x_50, r_x_50)
w_50 = np.dot(lin.inv(Rx), r_dx_50)

r_x_100 = r_x[1]
r_dx_100 = r_dx[1]

Rx = lin.toeplitz(r_x_100, r_x_100)
w_100 = np.dot(lin.inv(Rx), r_dx_100)

r_x_200 = r_x[2]
r_dx_200 = r_dx[2]

Rx = lin.toeplitz(r_x_200, r_x_200)
w_200 = np.dot(lin.inv(Rx), r_dx_200)

W = np.zeros((3,10))
W[0,:], W[1,:], W[2,:] = w_50, w_100, w_200

table = np.zeros((4,10))
table[0,:] = np.arange(10)
table[1:,:] = W

Headers = ["k","50cm","100cm","200cm"]
print("Table for")
print(tabulate(table.T, headers=Headers))
```

```
Table for
  k         50cm        100cm        200cm
---  -----------  -----------  -----------
  0   0.50181      0.14448      0.14368
  1   0.469675     0.598286     0.600635
  2   0.0833732    0.210784     0.213955
  3   0.0466878    0.0575841    0.0538982
  4  -0.0740747    0.0015143    0.00758647
  5  -0.0396878   -0.0295098   -0.0272674
  6  -0.0264527   -0.0119507   -0.0212785
  7   0.00981699  -0.00413636  -0.00891275
  8   0.00424727  -0.0206574   -0.0120283
  9   0.00458949  -0.00129652  -0.00511856
```

4.7 Question 7

Using the computed filter coefficients, compute an estimate of the denoised audio signals from the received ones. Plot together the denoised signal with the originally received one. Create one plot for every audio signal.

Use the *play_signal()* function to play the denoised signals - do you hear the difference between the originally received and the denoised one? **(3 points)**

Hint: the *scipy.signal.lfilter()* function can be useful when using the Wiener filter. You can compare your solution to the *scipy.signal.wiener()* function, which uses its own estimate of the noise instead of the noise measurement. Do you notice any differences?

4.7.1 Answer 7

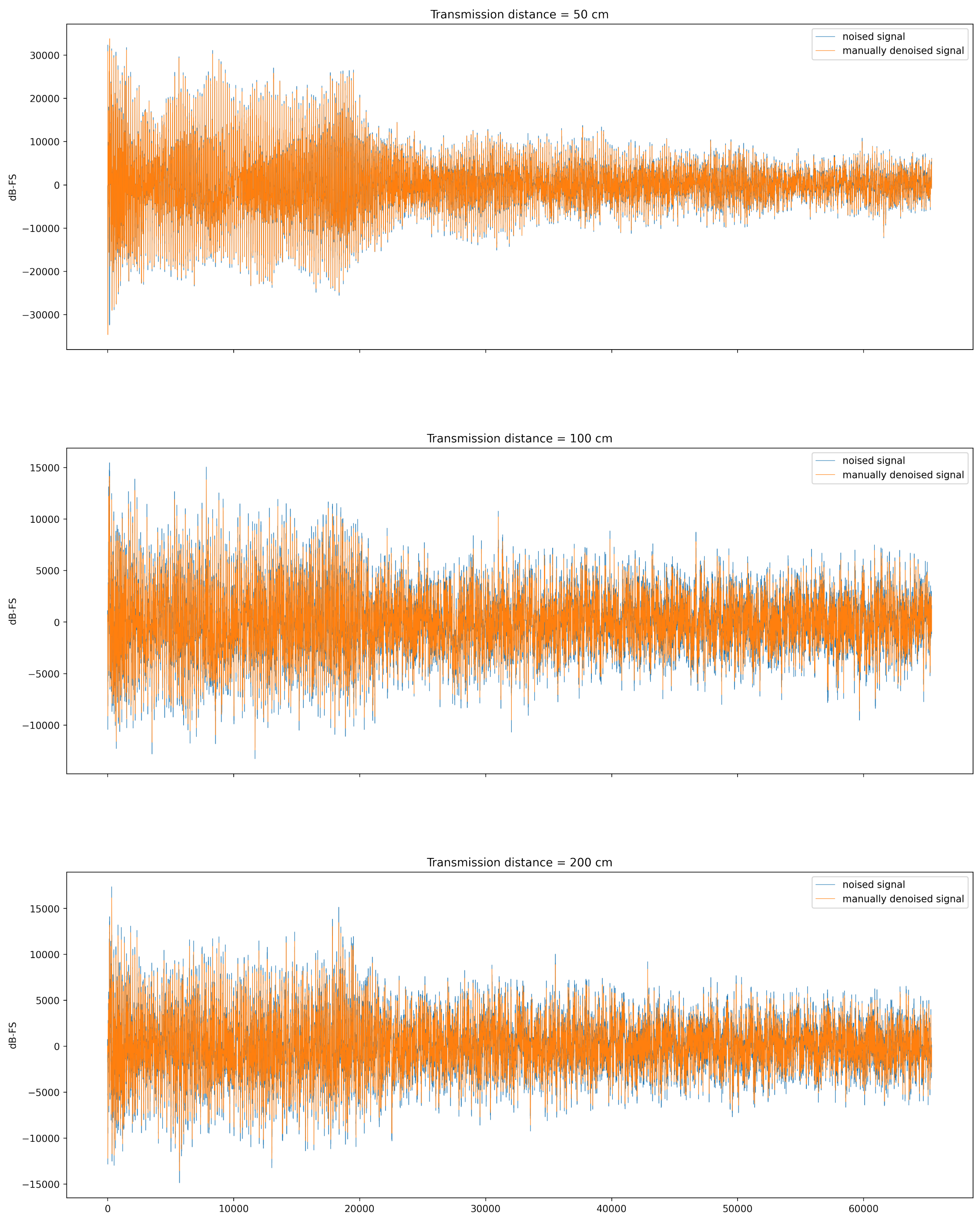
We can hear less noise but there is definitely still noise present. The Scipy function seems to do a better job at filtering.

[14]:
```
# applying the Wiener filter to the problem
import scipy.signal as sig

signal_denoised_50 = sig.lfilter(w_50, 1, data_noised[0])
signal_denoised_100 = sig.lfilter(W[1,:], 1, data_noised[1])
signal_denoised_200 = sig.lfilter(W[2,:], 1, data_noised[2])

# compare Scipy Wiener function to manually determination of Wiener constants
w_wiener_50 = sig.wiener(data_noised[0], 10)
w_wiener_100 = sig.wiener(data_noised[0], 10,)
w_wiener_200 = sig.wiener(data_noised[0], 10,)
```

[15]:
```
# listen the noised and denoised audio signals
import time

# index of the experiment that we want to hear
ind_to_hear = 2

# signal with noise
#play_signal( data_noised[0] )

# timer to wait for some time between sounds
time.sleep(3)

# denoised signal
#play_signal(signal_denoised_50)
```

[16]:
```
# plot the noised and denoised signals together
# plot the noised and denoised signals together
fig,ax = plt.subplots(3,1, sharex=True, dpi=250)
plt.tight_layout()
fig.set_figwidth(15)
fig.set_figheight(20)

#plot for distance of 50 cm:
```

```
ax[0].plot(data_noised[0], label= 'noised signal', linewidth=0.5)
ax[0].plot(signal_denoised_50, label= 'manually denoised signal', linewidth=0.5)
#ax[0].plot(denoised_50_sci, label= 'Scipy denoised signal', linewidth=0.5)
ax[0].legend()
ax[0].set_ylabel('dB-FS')
ax[0].set_title('Transmission distance = 50 cm')

#plot for distance of 100 cm:
ax[1].plot(data_noised[1], label= 'noised signal', linewidth=0.5)
ax[1].plot(signal_denoised_100, label= 'manually denoised signal', linewidth=0.
 ↪5)
#ax[1].plot(denoised_100_sci, label= 'Scipy denoised signal', linewidth=0.5)
ax[1].legend()
ax[1].set_ylabel('dB-FS')
ax[1].set_title('Transmission distance = 100 cm')

#plot for distance of 200 cm:
ax[2].plot(data_noised[2], label= 'noised signal', linewidth=0.5)
ax[2].plot(signal_denoised_200, label= 'manually denoised signal', linewidth=0.
 ↪5)
#ax[2].plot(denoised_200_sci, label= 'Scipy denoised signal', linewidth=0.5)
ax[2].legend()
ax[2].set_ylabel('dB-FS')
ax[2].set_title('Transmission distance = 200 cm')
plt.show()
```



4.8 Question 8

1. To investigate the effect of the filter order on the denoising, vary the filter order between $m = 2$ and $m = 20$. For what distance and for what filter order do you have the best denoised audio signal? Justify your answer. **(1 point)**

2. Denote the best denoised signal from item "a." as $d(n)$, and calculate the cost function J of the Wiener problem assuming this signal is the ground truth. Print best filter order and the its respective cost function value for every received denoised audio signal. **(1 point)**
3. Plot the filter order (from $m = 2$ to $m = 20$) versus cost function value for all received signals. **(1 point)**

4.8.1 Answer 8

We suspect that the best filtereing is for the highest order Wiener filter for the largest distance because the noise to data ratio is the highest. So its easiest to filter the noise.

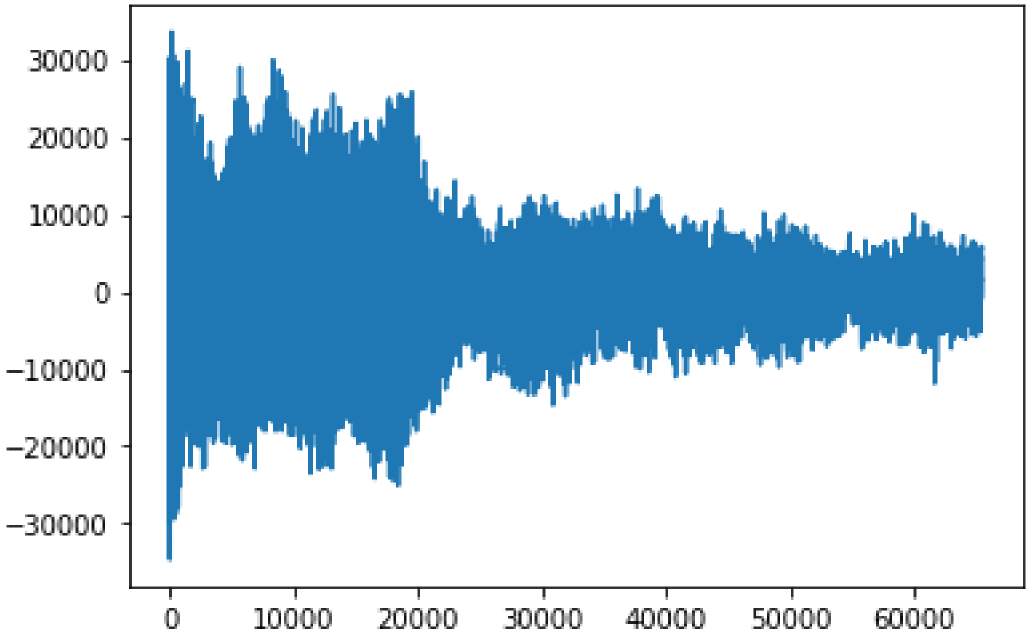
[17]:
```python
# use different Wiener filter orders
#For the 50cm signal
x = data_noised[0]
r_x = np.zeros(20)
r_v = np.zeros(20)
r_dx = np.zeros(20)

N1 = len(x)
N2 = len(noise_measured)


for i in range(20):
    k = i
    r_x[i] = 1/(N1-k) *np.sum(x[k:]*x[:(N1-k)])
    r_v[i] = 1/(N2-k) *np.sum(noise_measured[k:]*noise_measured[:(N2-k)])
    r_dx[i] = r_x[i]- r_v[i]


Rx = lin.toeplitz(r_x, r_x)
WW = np.dot(lin.inv(Rx), r_dx)


#Signals, Matrix of 18 rows where row+2 filters are used

Signals_50 = np.zeros((19,len(x)))
for i in range(2,21,1):
    Signals_50[i-2,:] = sig.lfilter(WW[0:i+1], 1, x)


#For the 100cm signal
x = data_noised[1]
r_x = np.zeros(20)
r_v = np.zeros(20)
r_dx = np.zeros(20)

N1 = len(x)
```

```
N2 = len(noise_measured)


for i in range(20):
    k = i
    r_x[i] = 1/(N1-k) *np.sum(x[k:]*x[:(N1-k)])
    r_v[i] = 1/(N2-k) *np.sum(noise_measured[k:]*noise_measured[:(N2-k)])
    r_dx[i] = r_x[i]- r_v[i]


Rx = lin.toeplitz(r_x, r_x)
WW = np.dot(lin.inv(Rx), r_dx)


#Signals, Matrix of 18 rows where row+2 filters are used

Signals_100 = np.zeros((19,len(x)))
for i in range(2,21,1):
    Signals_100[i-2,:] = sig.lfilter(WW[0:i+1], 1, x)




# use different Wiener filter orders
#For the 200cm signal
x = data_noised[2]
r_x = np.zeros(20)
r_v = np.zeros(20)
r_dx = np.zeros(20)

N1 = len(x)
N2 = len(noise_measured)


for i in range(20):
    k = i
    r_x[i] = 1/(N1-k) *np.sum(x[k:]*x[:(N1-k)])
    r_v[i] = 1/(N2-k) *np.sum(noise_measured[k:]*noise_measured[:(N2-k)])
    r_dx[i] = r_x[i]- r_v[i]


Rx = lin.toeplitz(r_x, r_x)
WW = np.dot(lin.inv(Rx), r_dx)


#Signals, Matrix of 18 rows where row+2 filters are used

Signals_200 = np.zeros((19,len(x)))
```

```
for i in range(2,21,1):
    Signals_200[i-2,:] = sig.lfilter(WW[0:i+1], 1, x)

d = Signals_50[-1]
cost_w_50 = np.average( np.abs(Signals_50 - d)**2, axis=1 )
cost_w_100 = np.average( np.abs(Signals_100 - d)**2, axis=1 )
cost_w_200 = np.average( np.abs(Signals_200 - d)**2, axis=1 )
```

[18]:
```
# plot the best denoised signal, which we will take to be the ground truth
plt.plot(d)
```

[18]:
```
[<matplotlib.lines.Line2D at 0x7f11f146b460>]
```



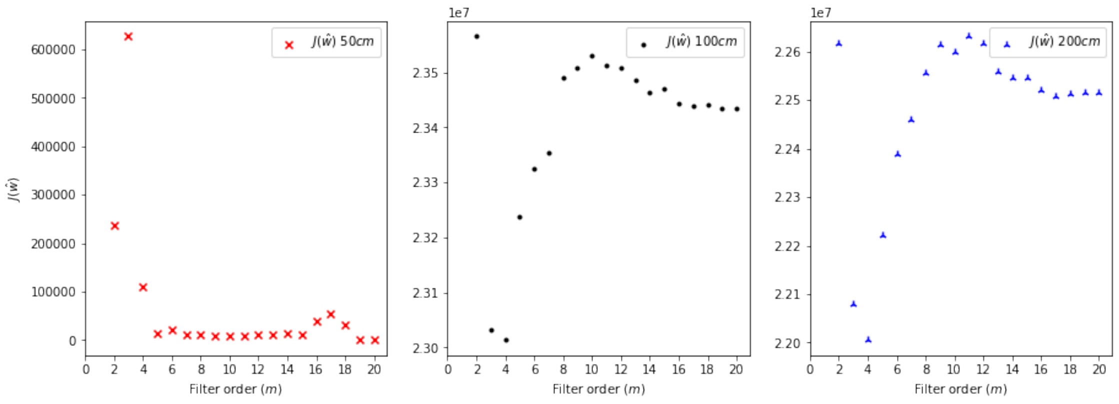
[19]:
```
# play the audio signal that is considered as ground truth
#play_signal(d)
```

[20]:
```
# calculating the cost function for the best signal
table = np.zeros((4,19))
table[0,:] = np.arange(19)
table[1,:], table[2,:], table[3,:] = cost_w_50, cost_w_100, cost_w_200

headers = ["50cm", "100cm", "200cm"]



display(Math("J(\hat{w})\; for \; d(n) \; is:"))
print(tabulate(table.T, headers=headers))
```

$J(\hat{w})$ *for* $d(n)$ *is* :

```
        50cm          100cm         200cm
--  ---------  -----------  -----------
 0  237588     2.35656e+07  2.26183e+07
 1  627114     2.30316e+07  2.20803e+07
 2  111395     2.30131e+07  2.2005e+07
 3   14110.6   2.32373e+07  2.22229e+07
 4   21820.4   2.33244e+07  2.23901e+07
 5   12489.4   2.33546e+07  2.24609e+07
 6   10436.1   2.34905e+07  2.25573e+07
 7    9211.03  2.35092e+07  2.26137e+07
 8    8994.13  2.35315e+07  2.25993e+07
 9    9821.24  2.35122e+07  2.26317e+07
10   10738.1   2.35084e+07  2.26173e+07
11   10266.8   2.34869e+07  2.25593e+07
12   13758.9   2.34629e+07  2.25473e+07
13   12483.3   2.347e+07    2.25458e+07
14   38151.1   2.34432e+07  2.25203e+07
15   55179.9   2.34393e+07  2.25086e+07
16   31053.1   2.34405e+07  2.25127e+07
17       0     2.34345e+07  2.2515e+07
18       0     2.34345e+07  2.2515e+07
```

[21]:
```
# plot cost functions as a function of the filter order
xaxis = np.arange(19)+2
yaxis_50 = cost_w_50
yaxis_100 = cost_w_100
yaxis_200 = cost_w_200

fig, axs = plt.subplots(1, 3)
plt.gcf().set_size_inches(15,5)
axs[0].scatter(xaxis, yaxis_50, marker="x", color='red', label=r"$J(\hat{w})$␣
 ↪$50cm$")
axs[1].scatter(xaxis, yaxis_100, marker=".", color='black',␣
 ↪label=r"$J(\hat{w})$ $100cm$")
axs[2].scatter(xaxis, yaxis_200, marker="2", color='blue',label=r"$J(\hat{w})$␣
 ↪$200cm$")
axs[0].legend()
axs[1].legend()
axs[2].legend()
axs[0].set_xticks(np.arange(11)*2);
axs[1].set_xticks(np.arange(11)*2);
axs[2].set_xticks(np.arange(11)*2);
axs[0].set_ylabel(r"$J(\hat{w})$");
axs[0].set_xlabel(r"Filter order $(m)$");
axs[1].set_xlabel(r"Filter order $(m)$");
axs[2].set_xlabel(r"Filter order $(m)$");
```



4.9 Question 9 (bonus)

This is an **optional** bonus question, in which you can earn bonus points to make up for any points you lost earlier during this exercise. Note that if you already have the maximum amount of points for this exercise (i.e. grade 10), any additional bonus points will be ignored (that is, they do not carry over to other exercises or the exam).

In this bonus question, we will characterize the signal over distance dependency by computing the signal-to-noise-ratio (SNR). For that: 1. Calculate the periodogram P_d using the denoised audio signals only for the highest filter order from Question 8. Calculate the periodogram P_x using the noised audio signals (see section "*5.2.8 WSS Processes in the Frequency Domain*" in the lecture notes). Plot the **denoised** estimated power spectra. **(2 bonus points) Hint:** be careful with the indexing. You can use *numpy.fft.fft()* and *numpy.fft.fftfreq()* functions.

2. SNR is the ratio between the **ground truth** audio signal energy and the noise energy. The energy can be calculated as $E = \int_{-\infty}^{+\infty} P d\omega$. Approximating the integral by finite sum, calculate

   (a) the SNR of the denoised audio signals;

   (b) the SNR of the noised audio signals. This can be done by relating the power spectrum of the noised signal with the power spectrum of the ground truth signal.

   In a single figure, plot these SNR values versus distance. **(2 bonus points)**

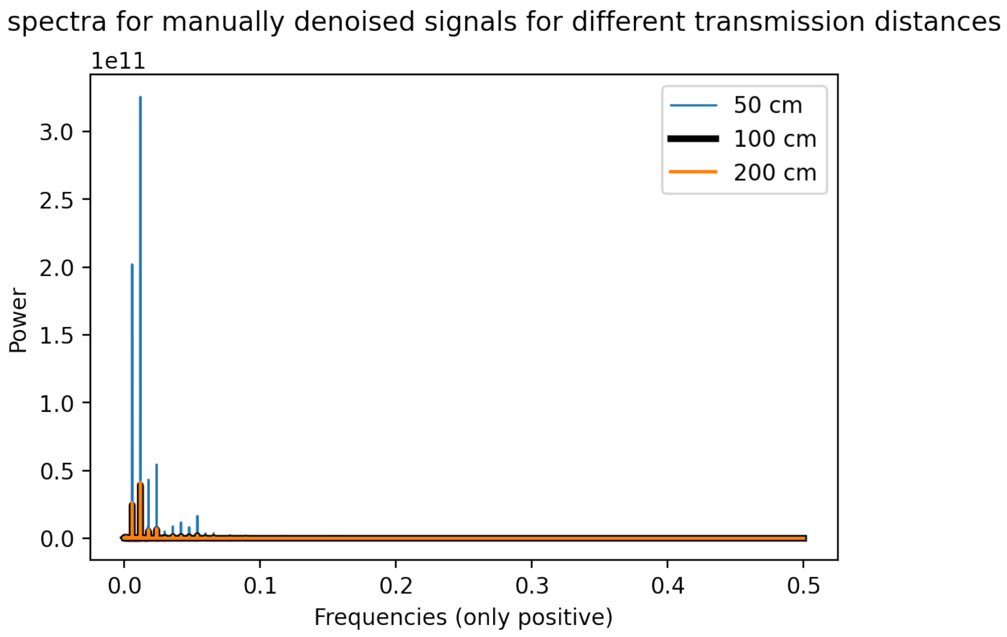
3. Are the graphs different? Explain why. **(1 bonus point)**

4.9.1 Answer 9

We can see that the graphs are not the same, we assume this is because the filtered signal $d(n)$ has a lower Power spectrum since we filtered its noise (the power from the noise doesnt contribute anymore). Since we still divide it by the power of the pure noise the fraction is lower than that of the unfiltered signal that still has power from the white noise.

```
[22]: # calculating the periodogram for denoised audio signals
      # and noised audio signals
      # (estimation of the power spectrum)

      import scipy.integrate as integrate
      import numpy.fft as fft

      # some usefule parameters
      x50 = data_noised[0]
      x100 = data_noised[1]
      x200 = data_noised[2]

      d50 = Signals_50[18,:]
      d100 = Signals_100[18,:]
      d200 = Signals_200[18,:]

      L = len(x50)
      freq_unshifted = fft.fftfreq(L)
      freq = fft.fftshift(fft.fftfreq(L))


      # calculating the periodogram for denoised audio signals
      D50 = fft.fft(d50)
      P_D50 = 1/(len(d50))*(D50*np.conj(D50))
      E_D50 = integrate.simps(P_D50,freq)

      D100 = fft.fft(d100)
      P_D100 = 1/(len(d100))*(D100*np.conj(D100))
      E_D100 = integrate.simps(P_D100,freq)

      D200 = fft.fft(d200)
      P_D200 = 1/(len(d200))*(D200*np.conj(D200))
      E_D200 = integrate.simps(P_D200,freq)

      # Calculate periodograms for noised audio signals

      X50 = fft.fft(x50)
      P_X50 = 1/(len(x50))*(X50*np.conj(X50))
      E_X50 = integrate.simps(P_X50,freq)


      X100 = fft.fft(x100)
      P_X100 = 1/(len(x100))*(X100*np.conj(X100))
      E_X100 = integrate.simps(P_X100,freq)

      X200 = fft.fft(x200)
      P_X200 = 1/(len(x200))*(X200*np.conj(X200))
      E_X200 = integrate.simps(P_X200,freq)
```

```
[23]: # plot power spectrum values
      # plot power spectrum values
      import matplotlib
      matplotlib.rcParams["figure.dpi"] = 200

      plt.plot(freq_unshifted[0:int(len(P_D50)/2)], P_D50[0:int(len(P_D50)/2)],␣
       ↪label= '50 cm', linewidth=1);
      plt.plot(freq_unshifted[0:int(len(P_D100)/2)], P_D100[0:int(len(P_D100)/
       ↪2)],color="black", label= '100 cm', linewidth=3);
      plt.plot(freq_unshifted[0:int(len(P_D200)/2)], P_D200[0:int(len(P_D200)/2)],␣
       ↪label= '200 cm', linewidth=1.5);
      plt.title('Power spectra for manually denoised signals for different␣
       ↪transmission distances \n');
      #plt.xlim([0,0.1])
      #plt.semilogy()
      plt.xlabel('Frequencies (only positive)')
      plt.ylabel('Power')
      plt.legend();
      plt.show()
      plt.tight_layout()
```

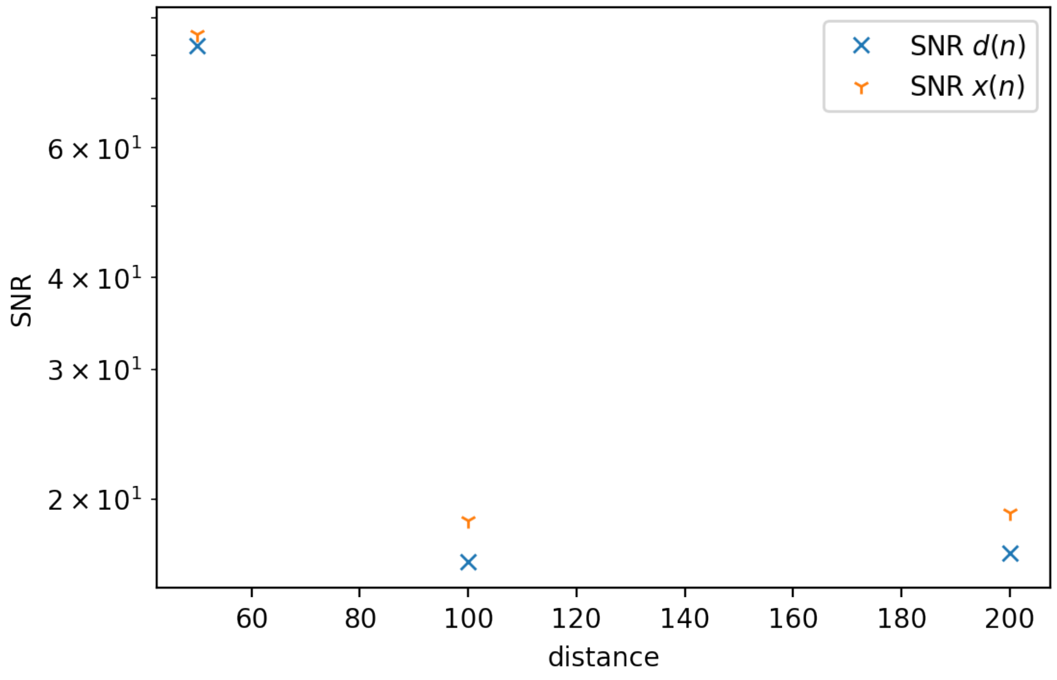


```
<Figure size 1200x800 with 0 Axes>
```

```
[24]: # plot the SNR over distance
      #define noise as e(n) = d(n) - \hat{d}(n)
```

```
Noise = fft.fft(noise_measured)
Noise_freq = fft.fftfreq(len(noise_measured))
PNoise = 1/(2*len(noise_measured)+1)*(Noise*np.conj(Noise))
E_noise = integrate.simps(fft.fftshift(PNoise),fft.fftshift(Noise_freq))

SNR_D = np.asarray([E_D50, E_D100, E_D200])/E_noise
SNR_X = np.asarray([E_X50, E_X100, E_X200])/E_noise
```

[25]:
```
xdist = [50,100,200]

plt.plot(xdist, SNR_D, label=r"SNR $d(n)$", linestyle='none', marker='x')
plt.plot(xdist, SNR_X, label=r"SNR $x(n)$", linestyle='none', marker='1')
plt.legend()
plt.semilogy()
plt.xlabel("distance");
plt.ylabel("SNR")
plt.show()
```


SNR $d(n)$
SNR $x(n)$
6×10^1
4×10^1
3×10^1
2×10^1
SNR
distance