CPE 4040: Homework 4

1. Write your answer in the cell provided under each question.
2. **You must write comments to explain your thoughts and earn full credit**.
3. **Show your execution result**.
4. Do your own work. **Do not copy-and-paste other people's (or Generative AI's) codes.**
5. You can find Matplotlib code snippets in the accompanying Jupyter Notebook file to help you make time-domain and frequency-domain plots.

Submission:

- **Submit this notebook file and the pdf version** - remember to add your name in the filename.

In [195…
```
import pandas as pd
import numpy as np

from numpy.fft import fft, ifft
```

In [195…
```
import matplotlib.pyplot as plt
```

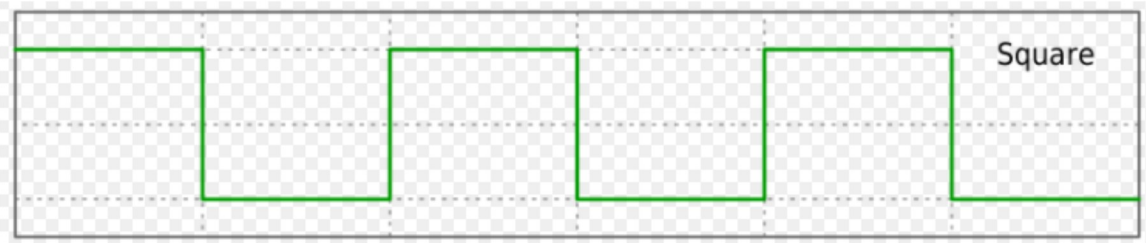
Q1: Signal Processing: Square Wave and Random Noise (60 Points)

A communication system is commonly modelled by the transmitted signal, the channel, and the receiver.

Let's consider a simple system where the signal is a `square wave` and the the channel adds `white Gaussian noise` to the signal. Therefore, the received signal is represented by

$$r(t) = s(t) + n(t)$$

where $s(t)$ = square wave, $n(t)$ = Gaussian noise.



1. (8 Points) Generate a periodic square wave by taking a `sign`

`function` on the sine wave, that is,

$$s(t) = \begin{cases} 1 & \text{if } \sin(2\pi ft) \geq 0 \\ -1 & \text{if } \sin(2\pi ft) < 0 \end{cases}$$

- Plot a 10-second **square wave** with amplitude 1, f = 4Hz and sampling rate f_s = 100Hz. Use Matplotlib to plot the signal. You should have a total of 1000 samples.
- Properly label the x-axis ("Time"), and y-axis ("Amplitude").

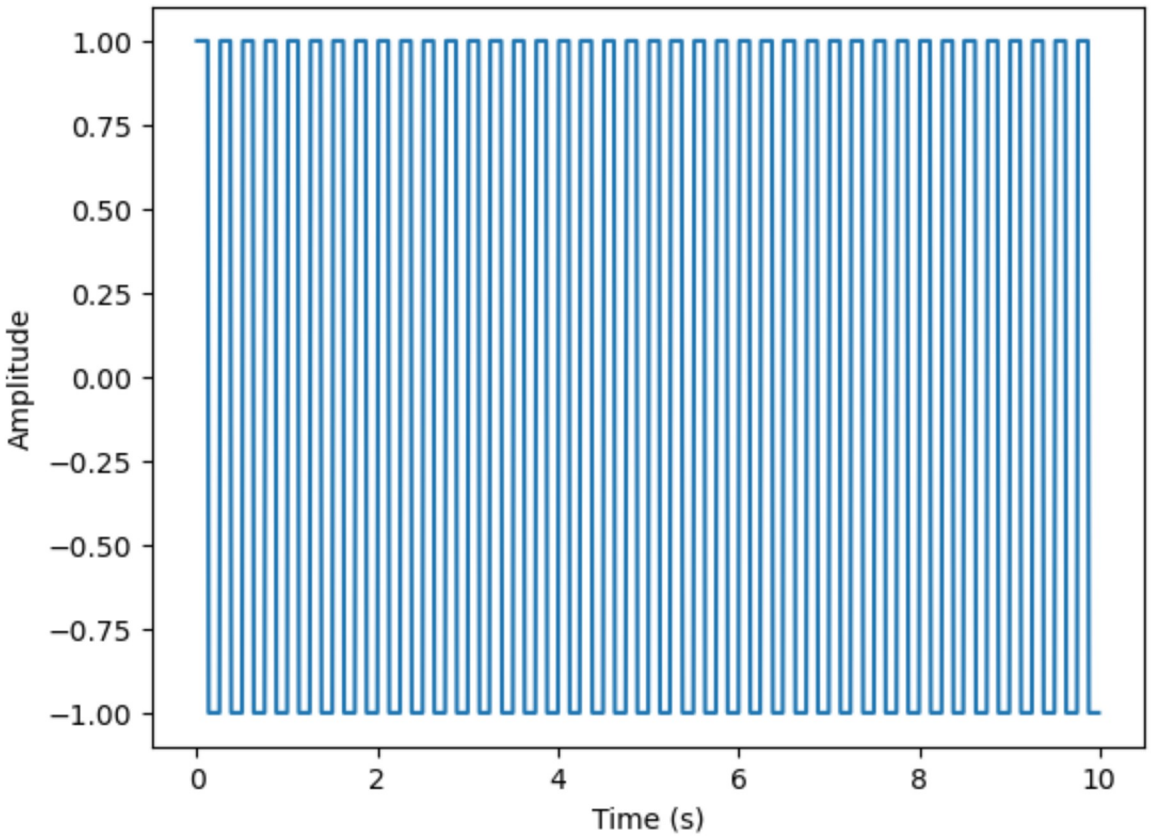
In [195…
```
def generate_sine_wave(freq: int, sample_rate:int, duration: int) -> np.ndarray | n

    t = np.linspace(0, duration, sample_rate * duration, endpoint=False)
    y = np.sin((2 * np.pi) * freq * t)

    return t, y
```

Generate the square wave

In [195…
```
def sign_function(s):
    if s >= 0:
        return 1
    if s < 0:
        return -1

def square_wave(freq: int, sample_rate:int, duration: int)  -> np.ndarray | np.ndar
    t, y = generate_sine_wave(freq, sample_rate, duration)

    y = np.array([sign_function(x) for x in y])

    return t, y


freq = 4#Hz
sampling = 100#Hz
duration = 10#s

t, y = square_wave(freq, sampling, duration)
```

Plot the square wave

In [196…
```
fig, ax = plt.subplots()
ax.set_xlabel("Time (s)")
ax.set_ylabel("Amplitude")
ax.plot(t, y)
plt.show()
```



2. (6 Points) Frequency domain representation of the square wave.

Apply the `calculate_fft( )` function below and use Matplotlib to plot the Fourier Transform result for `frequency range from 0 to 50 Hz` .

In [196…
```
def calculate_fft(signal,fs):
    N = len(signal)
    n = np.arange(N)
    freqs = n*fs/N                # the frequency bins
    mag_fft = abs(fft(signal)/N)   # calculate the magnitude of fft
    return freqs, mag_fft
```

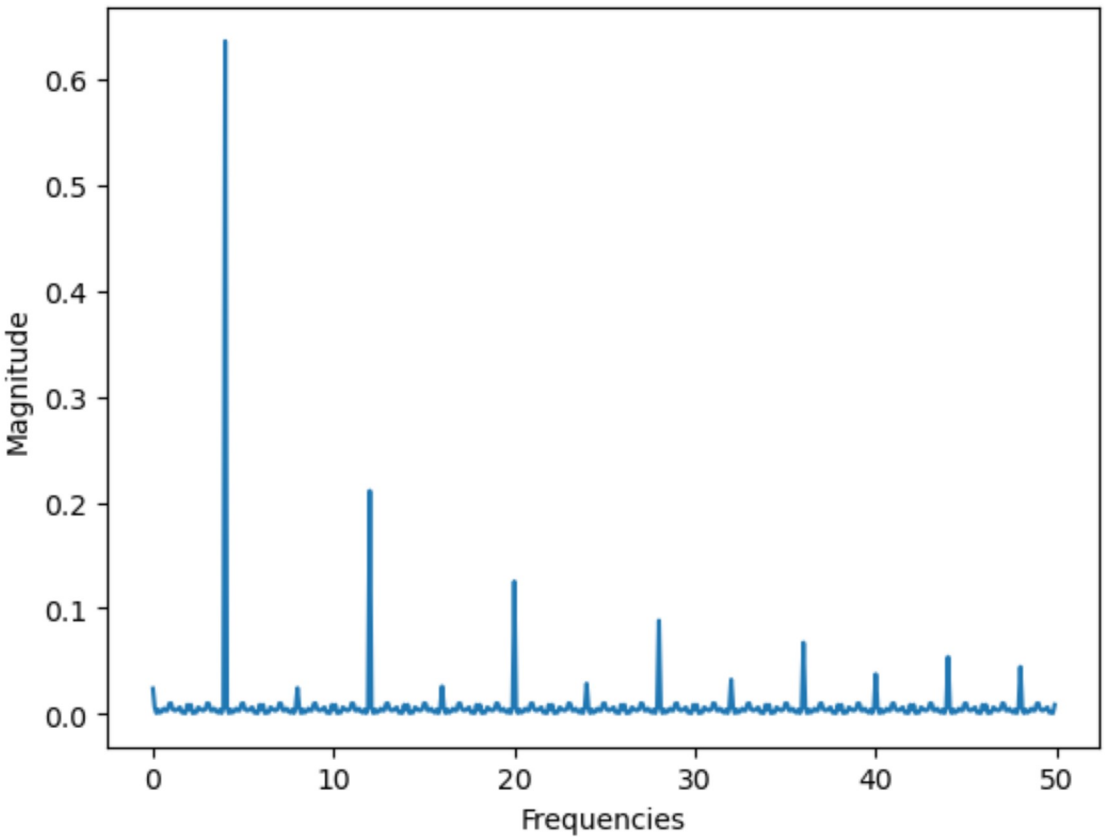
Calculate the FFT

In [196…
```
freqs, mag_fft = calculate_fft(y, sampling)
```

Plot FFT of the square wave for frequency range 0 to 50Hz

In [196…
```
mask = freqs < 50

filtered_mags = mag_fft[mask]
filtered_freqs = freqs[mask]

fig, ax = plt.subplots()
```

```
ax.set_xlabel("Frequencies")
ax.set_ylabel("Magnitude")
ax.plot(filtered_freqs, filtered_mags)
plt.show()
```



3. (10 Points) Find Peak Frequency Components

You can see a few distinct peaks in the FFT from Step 2. Write a code to identify the frequencies of the top five peaks in the FFT plot, `for frequency range from 0 to 50 Hz` .

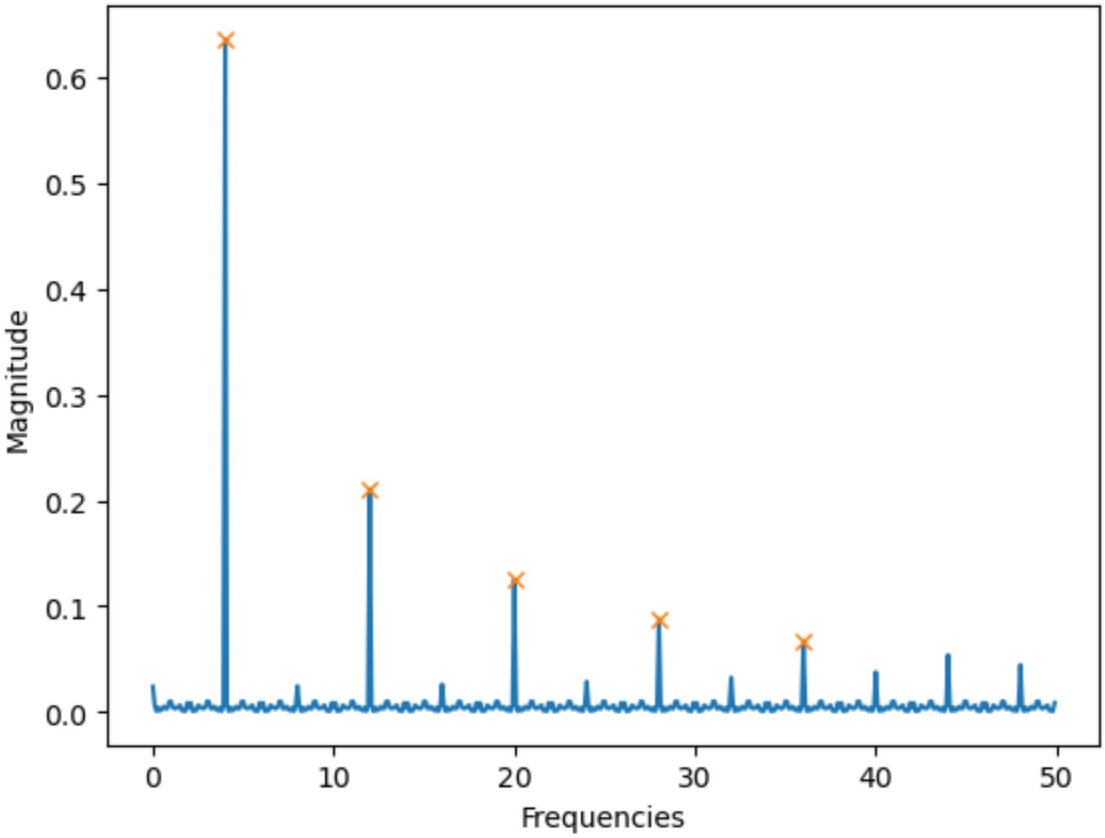
Please list the frequencies and their corresponding magnitudes.

In [196…

```
from scipy.signal import find_peaks



peaks, _ = find_peaks(filtered_mags)

peak_mags = filtered_mags[peaks]
top_5_mags = peaks[np.argsort(peak_mags)[-5:]]

print(filtered_freqs[top_5_mags])

fig, ax = plt.subplots()
ax.set_xlabel("Frequencies")
ax.set_ylabel("Magnitude")
ax.plot(filtered_freqs, filtered_mags)
```

```
ax.plot(filtered_freqs[top_5_mags], filtered_mags[top_5_mags], "x")
plt.show()
```

```
[36. 28. 20. 12.  4.]
```



4. (5 Points) Generate an array of Gausian noise with mean = 0 and standard deviation = 0.5

Use `np.random.normal` to generate the noise array. The size of the noise array should also be 1000.

Add the square wave array from 1 and the noise array together. Plot the resulting array.

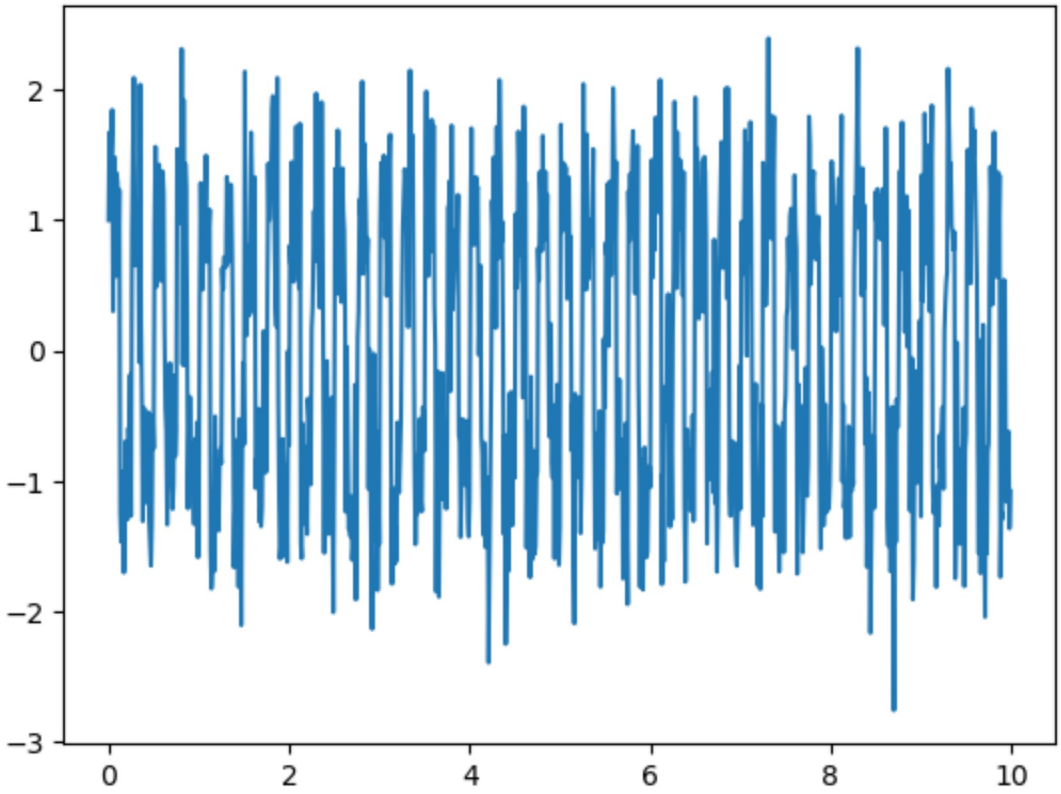
Generate the Gaussian noise and the noisy signal

In [196…

```
def square_with_noise(freq: int, sample_rate:int, duration: int) -> np.ndarray | np
    t, y = square_wave(freq, sample_rate, duration)
    noise = np.random.normal(0, 0.5, t.size)
    y = np.add(y, noise)

    return t, y

freq = 4#Hz
sampling = 100#Hz
duration = 10#s

t, y_noise = square_with_noise(freq, sampling, duration)
```

Plot the noisy signal

In [196…
```
fig, ax = plt.subplots()
ax.plot(t, y_noise)
plt.show()
```



5. (6 Points) Signal-to-noise ratio (SNR) is an important quality indicator of the communication channel.

```
 SNR(dB) = 10log10(P_s/P_n), where P_s is the average signal power
and P_n is the average noise power.
```

Please calculate the SNR(dB) of this case. The average power of a sequence of signals = $\sum_{i=1}^{N} s^2/N$, where N is the number of samples.

In [196…
```
def P_n(signal: np.ndarray) -> float:
    square = np.multiply(signal, signal)
    return np.average(square)

def snr (signal: np.ndarray) -> np.ndarray:
    pn = P_n(signal)
    ratio = signal / pn
    return 10 * np.log10(ratio)

snr(y_noise)
```

```
C:\Users\rbrin\AppData\Local\Temp\ipykernel_35876\2945329666.py:8: RuntimeWarning: i
nvalid value encountered in log10
  return 10 * np.log10(ratio)
```

Out[196…

```
array([-9.86826686e-01,  1.25461371e+00,  4.09354932e-01,  9.93391567e-01,
        1.68002916e+00, -6.20584656e+00,  4.52931715e-01,  7.24576773e-01,
       -3.43107490e+00,  3.51228138e-01,  9.72446916e-02, -2.44878741e+00,
       -8.25944928e-02,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan, -1.39396272e+00,  6.51182686e-01,
        2.22415881e+00,  5.45221117e-01, -2.84271340e+00, -1.22042094e+00,
       -2.18802204e+00, -1.43767951e+00,             nan,  2.11464731e+00,
       -7.36223466e+00,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan, -1.04192496e+00,
        9.55645305e-01, -4.10368945e+00, -1.06293970e+00,  2.04321258e-01,
        5.58698296e-01,  1.26123088e-01, -3.70380459e+00,  3.86067316e-01,
        3.95529792e-01, -1.68980375e-01, -8.91607308e+00,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
        9.06314280e-01, -8.01945924e-01, -1.07970899e+00,  2.30520243e-01,
       -4.36127985e-01,  2.66114589e+00,             nan,  1.85793822e+00,
                   nan,  5.81115319e-01, -3.07089737e-02, -5.79304516e+00,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan, -5.26845840e+00,  1.07046083e-01, -2.19568860e+00,
       -4.30403275e+00, -2.43799285e+00, -6.20619775e-02, -3.49439880e-01,
        7.68586029e-01, -2.66251736e+00, -1.80375011e+00, -1.32684281e+00,
       -6.43605035e-01,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan, -2.98050311e+00, -4.26141316e+00,
       -2.43754297e+00, -2.46889696e+00, -2.91789763e+00,  2.61000473e-01,
       -1.95049309e-01, -2.67406182e+00, -1.30759991e+00,  6.90492990e-02,
        3.37060144e-02, -2.44493091e+00,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,  2.32122187e+00,
       -9.19671844e+00, -1.03869397e+01, -4.73149429e+00, -1.96969828e+00,
       -6.68637753e+00, -5.76123661e-01,  1.25242030e+00,  1.42248089e-01,
       -6.14165057e+00, -2.88311347e+00,  2.52753850e-01,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
       -9.23632175e+00,             nan,             nan,             nan,
       -4.31835240e+00,  5.86244400e-01, -9.82361410e-01,  4.76699365e-02,
        2.95521994e-01,  1.68623740e+00,  1.92184001e+00, -2.73098910e-02,
       -1.73284798e+00, -5.45095821e+00, -8.32265098e+00,  2.22251771e+00,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan, -1.91221242e+00, -2.78086830e+00,  6.18856643e-01,
       -6.38090067e-01, -3.76757195e+00, -1.03655815e+00, -2.14051200e+00,
        1.37107550e+00, -1.59474914e+00, -2.77000979e+00, -4.20060847e+00,
        1.42698796e+00,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
                   nan,             nan,             nan,             nan,
```

```
           nan, -8.79175623e+00, -5.87426713e+00, -2.22583174e+00,
-6.80073738e-01, -2.60691555e+00,  1.96803997e+00, -2.26414921e+00,
 1.83757326e-01, -4.23381260e+00, -5.83331029e+00, -2.16249020e+00,
 1.81840195e+00, -4.25152238e-01,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,  4.57355866e-01,
-4.68295081e-01, -5.40278055e-01,  1.29656279e+00, -4.61992383e+00,
-1.28057909e+00, -8.38040831e-01, -5.29074260e+00,  4.73198958e-01,
-6.88814193e-01, -1.30737868e+00, -6.14910039e+00,             nan,
-1.66129306e+01,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
-1.11570143e+01, -6.13633411e+00, -3.54136334e-01, -7.72079643e-01,
 4.40062259e-01,  2.16526862e+00, -3.30050620e+00,  1.03478229e+00,
 5.47780770e-01, -5.32012612e-01, -2.64517176e+00, -1.65304510e+00,
           nan, -2.07082946e+01,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan, -1.22010467e+00,  5.99167649e-01, -9.29031903e-01,
-4.77749057e-01,  7.69011320e-01, -1.61603428e+00, -2.34365428e-01,
-4.76486824e+00, -2.35554691e+00, -3.13484445e-01,  1.06354400e+00,
 1.21114645e+00,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan, -5.10804674e+00, -1.22558400e+00, -2.02886549e-01,
 4.54519619e-01,  6.82634392e-02,  1.26300204e-01, -2.06868773e+00,
-8.48349838e+00,  3.25979770e-01,  2.34023767e+00, -3.81050968e-01,
-2.84607203e-01,  1.19520305e+00, -4.57591118e+00,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,  4.21084631e-01,
 2.00107693e+00, -6.28128932e-01, -3.42364673e+00, -9.99829304e-01,
-8.76894953e-01,  8.83981480e-01,  1.49889611e+00, -2.18237233e+00,
 1.39234657e+00, -5.45286263e+00, -1.74241178e+01,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
-5.94393883e-01, -5.01608864e-01,  1.58474021e-01,             nan,
 1.38984052e+00, -6.05212594e+00, -4.30160535e+00, -2.45179694e+00,
-2.85608259e+00, -1.31554165e+00, -1.73212053e+00, -2.09867181e-01,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan, -1.13564060e+00,  1.33343222e+00, -1.49478495e+00,
-5.53137779e-01, -1.89995652e+00, -8.75460975e-01,  2.58998556e-01,
-6.38144767e-02,  2.52839188e-03,             nan, -6.58957344e+00,
-2.83604367e+00,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan, -3.71678298e-01, -1.27416416e+00,  7.31058649e-01,
-9.66827679e-01, -8.64033237e+00,  1.35789010e+00, -2.51973060e+00,
 6.57700692e-01,  2.18965451e+00,  1.30271777e+00, -1.80023924e+00,
-1.03140102e+00, -1.20092718e+01,             nan,             nan,
           nan,             nan,             nan,             nan,
           nan,             nan,             nan,             nan,
```

```
            nan,             nan,             nan, -7.83964912e-01,
-4.17821304e+00, -3.63776111e+00,  1.27057651e+00,  3.74666011e-01,
-2.33556988e+00, -1.44013352e+00, -1.06316881e+00,             nan,
 1.73846203e+00,  1.53658688e-01,  1.20275118e-01,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
-2.03171188e+00, -3.66538093e+00,  3.86005752e-01, -4.59388891e-01,
-2.15072856e+00,  1.18545391e+00, -1.38623959e+00, -1.69425736e+00,
 3.90163278e-01, -1.46339391e+00, -2.08245027e-01, -7.74480861e+00,
            nan,             nan, -7.88922053e+00,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,  1.40560416e+00, -1.38905669e+00, -6.42710919e-01,
 5.97011521e-01,  2.30132470e-01,  5.13309054e-01,  4.55778733e-01,
-5.00614621e+00, -5.55633050e-01,  2.45636226e-01, -3.43569322e+00,
-1.56840893e+00,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan, -2.43264242e+00,  2.12407253e+00, -1.39228539e+00,
-5.65716541e-01, -4.30841429e+00,  1.21450813e+00, -2.30876918e+00,
-3.58932260e+00, -9.93155449e-01, -1.89457150e+00, -1.74156167e+00,
-9.25467389e-01,  9.08140484e-01,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan, -1.14119831e+01,             nan, -1.81178384e+00,
-2.35104471e+00,  8.38769967e-02, -1.56895389e+01,  1.40312641e-01,
-8.11628642e-01, -3.35931640e+00, -6.13144243e-01,  2.05623685e+00,
 9.47202231e-01, -1.58861892e+00,  6.29957715e-01,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
-1.09996833e-01, -1.74688929e+00,  3.37402942e-01, -1.42063679e+00,
 7.01556419e-01,  1.27889464e+00, -1.03280880e+00, -4.55765193e+00,
 5.48162234e-01, -3.73782325e+00,  9.79968699e-01,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan, -6.48345023e-01,  6.68494535e-01, -3.50102262e+00,
 3.87444062e-01, -2.10355414e+00,  1.54002052e+00,  1.29059032e+00,
-6.14975543e-01, -7.33471059e-01,  1.27778376e+00,  2.19167681e+00,
 5.62940184e-01,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
-4.62420628e+00,             nan,             nan,             nan,
            nan, -9.56727468e+00, -3.05368338e+00,  1.82610726e+00,
-2.71822944e+00, -4.19597287e+00,  1.25989543e+00, -3.43840446e-01,
-1.30136735e+00,  6.66216024e-01, -8.61683362e-02, -2.39570969e+00,
-4.74346296e+00,  3.86131286e-01,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,  1.89849628e+00, -1.14341555e+00,
 9.17917233e-01, -7.73218793e-01, -7.13270762e+00, -8.29891414e-01,
-1.86519455e+00, -2.61986192e+00,  6.43167019e-01, -6.28398796e+00,
 7.37279859e-01,  3.81893561e-01, -1.73311837e+00,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan, -1.68333563e+00,
```

```
            nan,             nan,             nan,             nan,
-3.96230763e+00, -1.65727397e+00, -3.51549620e-01,  1.05828866e+00,
-1.31535819e+00, -2.98346657e+00, -1.29211416e+00, -3.51978318e-01,
 2.05330288e+00, -4.96449406e+00,  2.05872401e+00, -1.70086859e+00,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan, -1.04256562e+00, -3.32342025e+00, -2.90869506e+00,
-1.19462892e+00,  1.29768205e+00, -2.59738984e+00,             nan,
 7.97099145e-01,  1.00132802e+00, -2.61909372e+00,  1.45481461e+00,
-9.33278735e-01,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,  6.00424326e-01, -1.48187840e+00, -1.81593430e-01,
-5.60533916e+00, -2.70163350e+00,  9.29614305e-01,  2.81082627e+00,
-2.80598308e-01,  1.58766549e+00, -1.64438360e+00, -1.51578840e+00,
 4.42455156e-01,  1.55582967e+00,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan, -6.84813771e+00,
-5.80687346e+00, -1.60975237e+00, -2.97988887e+00, -8.41188688e-01,
-6.14969601e-01, -9.88980940e-01, -1.99879439e+01, -1.68078123e+00,
 3.01429838e-01, -1.23023947e+00, -2.10848507e+00,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
 1.55739932e+00,  9.94669200e-01, -3.91686735e+00, -1.44339941e+00,
 3.45505934e-02,  3.99969820e-01, -1.78320428e+00, -2.23322012e+00,
-2.55422834e+00, -1.39690377e+00, -8.58442599e-01, -2.37752122e+00,
            nan,             nan, -1.76978162e+01,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,  6.35509009e-01, -2.57643105e-01, -8.95385145e+00,
-5.50547593e-01, -2.14774581e+00, -9.36578849e+00, -2.03926495e+00,
-4.20788169e-01,  2.25141534e-01, -4.66571125e+00, -4.58319475e-01,
 1.57871916e+00,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan, -3.19321539e+00, -2.85804581e-01,
-5.89298133e-01,  1.46627443e+00,  2.66993002e+00, -4.45205792e-01,
-1.27537945e+00, -7.56895533e-01, -4.72046869e+00,  4.75500853e-01,
-4.28112487e+00, -5.02041466e-01,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan, -1.24235978e-01, -1.00135355e+00,
-5.06374505e-02, -9.81868641e-01, -2.60897562e-01, -1.66997471e+00,
-7.29809779e-01, -1.56300861e+00, -4.60290404e-02, -8.04668707e+00,
-7.85463025e+00,  1.34132338e+00, -2.13593853e+00,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
-1.22337509e+00,  4.14099419e-01, -1.22725363e+00,  1.44948232e+00,
-3.00426303e+00, -9.23209249e-01, -9.48201078e+00, -1.81543470e+00,
-2.47472623e-01, -1.68632317e+01, -6.64379122e-01, -5.60717197e+00,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
```

```
            nan,             nan,             nan, -7.36583589e+00,
            nan,  3.08680475e-01, -5.21233805e+00, -2.74517628e+00,
 1.61331073e+00,  3.36295955e-01, -5.63435471e-01, -2.75190375e+00,
 9.81837502e-01, -3.56301379e-01, -6.24865652e+00,  6.90196523e-01,
 1.76140564e+00,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan, -9.34955538e+00, -5.33128219e+00,
-3.81016940e+00, -3.17178639e+00,  2.36632297e+00,  1.44380440e+00,
 3.52201645e-01,  5.81778042e-01, -1.17401362e+00, -1.43852484e+00,
-2.09590247e+00, -1.39187214e+00,             nan,             nan,
-1.31852364e+01,             nan,             nan,             nan,
            nan,             nan,             nan,             nan,
            nan,             nan,             nan, -1.50171429e-01,
 8.96749541e-01, -2.84958902e-01, -3.36312029e+00, -3.87568699e+00,
 1.70755738e+00,  9.81738839e-01,  7.71604337e-01,  1.29719350e+00,
 2.41127945e-01, -8.28686204e-01, -3.63882869e+00,             nan,
            nan, -1.21161557e+01,             nan,             nan,
            nan, -8.04913078e+00,             nan,             nan,
            nan,             nan,             nan,             nan,
-3.21744539e+00,  5.13289564e-01, -6.15212926e-01, -5.52283890e+00,
 6.58561202e-01,  1.25553881e+00,  2.50640936e-01, -1.94117618e+00,
-3.47316674e+00,  3.82693754e-01, -1.56775409e+00,  3.03605327e-01,
            nan,             nan,             nan,             nan,
-3.64134850e+00, -4.62864120e+00,             nan,             nan,
            nan,             nan,             nan,             nan])
```

6. (10 Points) Implement a moving-average low-pass filter.

A moving-average filter is a simple yet effective digital low-pass filter that reduces random noises in the signal. It is characterized by its filter length L, specifying how many of the last points of the signal will be averaged to produce the next point in the output.

$$y[n] = \frac{1}{L} \sum_{i=0}^{L-1} x[n-i]$$

where $x[n]$ is the noisy signal, L is the length of the filter, and $y[n]$ is the filtered signal.

- Write a Python function `moving_average_filter` that implements this low-pass filter. The function should accept two parameters: `noisy_signal` and `L`. It should return the filtered signal array. For this exercise, set $L = 7$ for the length of the filter.
- Visualization: Plot the noisy signal and the filtered signal in the same figure.
- Please make a comment to compare the signals before and after the filtering.

Define the function for moving-average filter and calculate the filter output

In [196…

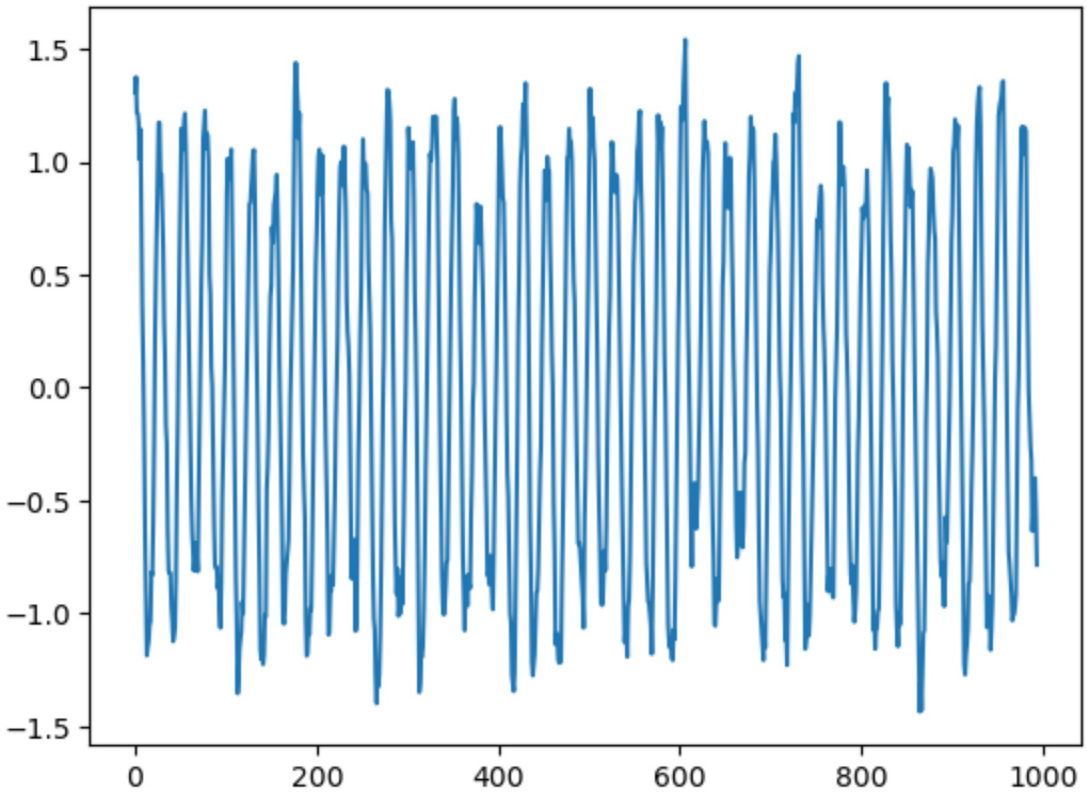
```
def moving_average_filter(noisy_signal: np.ndarray, L: int = 7) -> np.ndarray:
    filtered_signal = np.zeros(len(noisy_signal) - L + 1)
```

```
    for n in range(L - 1, len(y_noise)):
        filtered_signal[n - L + 1] = np.sum(noisy_signal[n - L + 1: n + 1]) / L

    return filtered_signal

low_pass = moving_average_filter(y_noise)
```

Plot the noisy signal and the filtered signal

In [196…

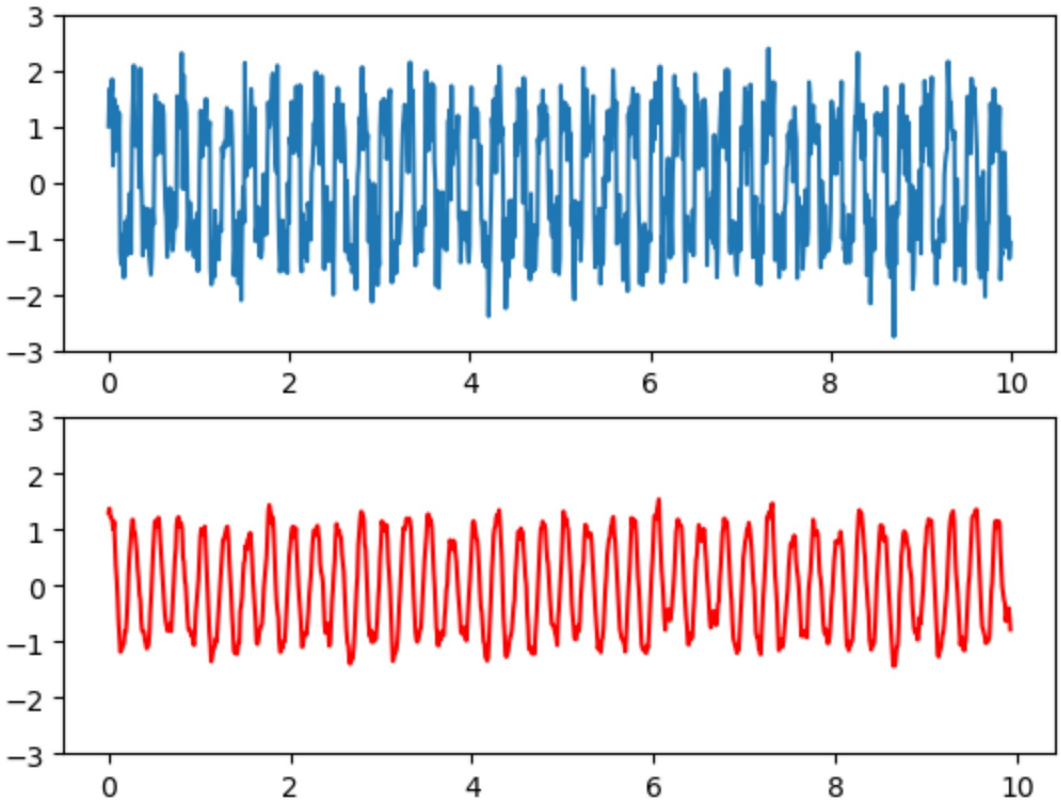
```
fig, ax = plt.subplots()
ax.plot(low_pass)
plt.show()
```



Compare the signals before and after the filtering.

In [197…

```
plt.subplot(2, 1, 1)
plt.plot(t, y_noise)
plt.ylim(-3, 3)

plt.subplot(2, 1, 2)
plt.plot(t[0:994], low_pass, 'r')
plt.ylim(-3, 3)
plt.show()
```



7. (8 Points) Frequency Domain view of the noisy signal and the filtered signal¶

Use FFT to transform both the noisy signal and the filtered signal to the frequency domain.

Visualization: Plot the noisy signal and the filtered signal in the same figure for comparison.You only need to plot in the frequency range between 0 to 5Hz.

Please make an observation on:

- The noise levels before and after moving-average filtering
- The peak frequency components: what happened there?

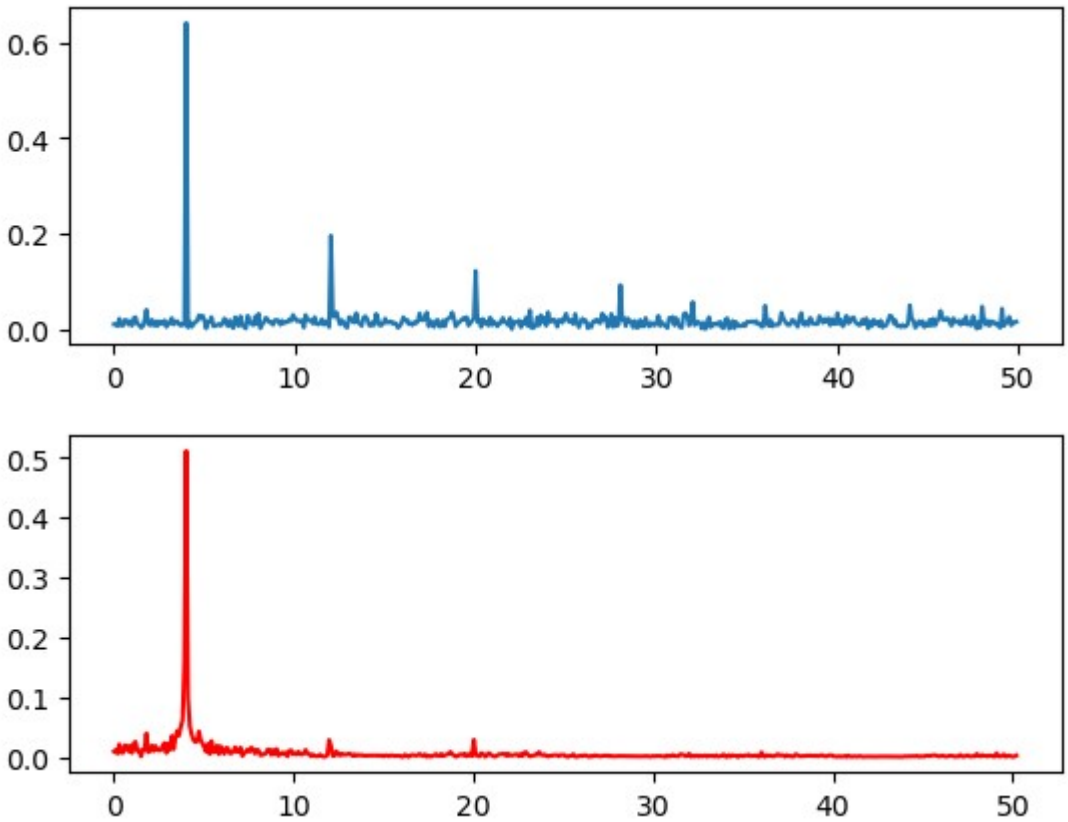
Calculate the FFTs

In [197…
```
freq_original, mag_original = calculate_fft(y_noise, sampling)
freq_low_pass, mag_low_pass = calculate_fft(low_pass, sampling)
```

Plot FFTs for both the noisy signal and the filtered signal, for frequency range from 0 to 50Hz

In [197…
```
mask = freqs < 50

filtered_mags_original = mag_original[mask]
filtered_freqs_original = freq_original[mask]
```

```
mask_low = freqs[:994] < 50

filtered_mags_low = mag_low_pass[mask_low]
filtered_freqs_low = freq_low_pass[mask_low]

plt.subplot(2,1,1)
plt.plot(filtered_freqs_original, filtered_mags_original)
plt.show()

plt.subplot(2,1,2)
plt.plot(filtered_freqs_low, filtered_mags_low, 'r')
plt.show()
```



Your observations

It appears that the low-pass filter worked. There is only one major peak and it is in the lower frequency range.

8. (7 Points) Frequency domain representation of the moving-average filter¶

To gain an insight on how the signal and the noise are shaped by the moving-average filter in the frequency domain, you will calculate the FFT of the filter (we call it the `frequency response`).

Recall in Step 6, the L-tap moving-average filter is represented by

$$h = [\frac{1}{L}, \frac{1}{L}, \frac{1}{L}, \cdots, \frac{1}{L}]$$

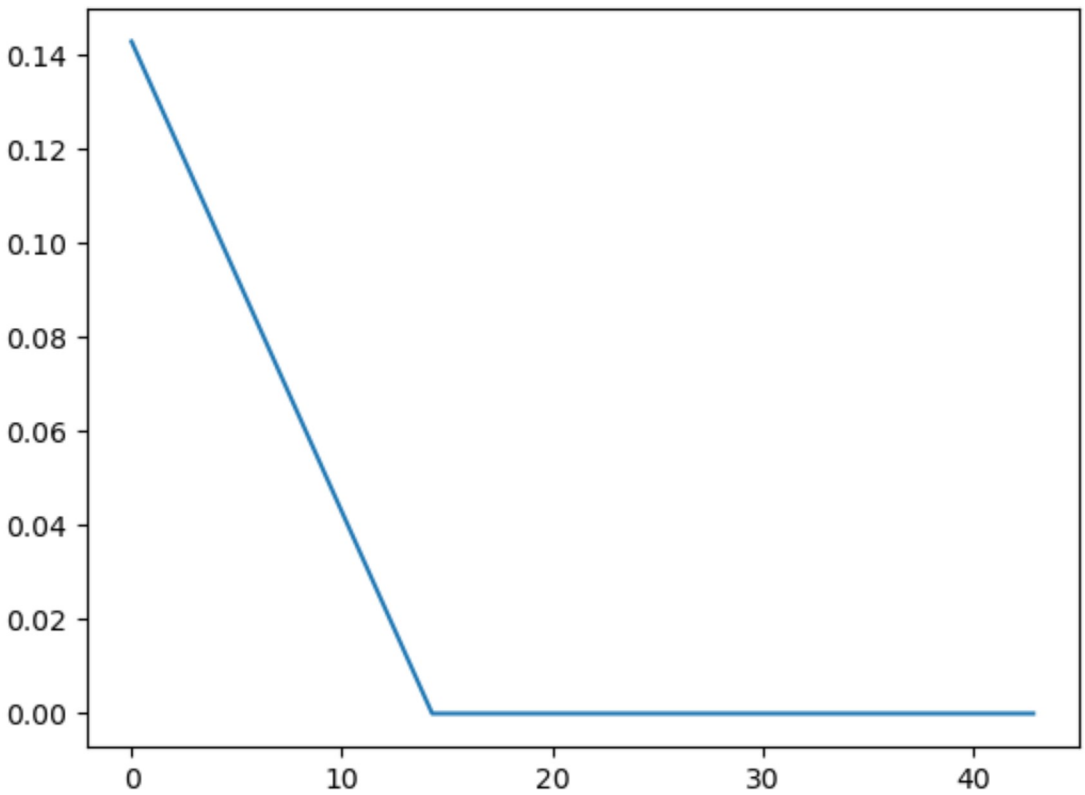
- Apply FFT on the filter, h, and plot the result for frequency range from 0 to 50Hz. Assume $L = 7$.
- Please make a comment on the shape of the frequency response and explain how it affects the noise and signal in Step 7.

Calculate the FFT of the 7-tap moving-average filter

In [197…
```
L = 7
h = np.ones(L) / L


freqs_h, mag_fft_h = calculate_fft(h, sampling)
```

Plot FFT of the 7-tap moving-average filter

In [197…
```
mask = freqs_h < 50

freqs_h_50 = freqs_h[mask]
mag_fft_h_50 = mag_fft_h[mask]

plt.plot(freqs_h_50, mag_fft_h_50)
plt.show()
```



Make a comment on the shape of the frequency response and explain how it affects the noise and signal in Step 7

The low-pass filter only lets low frequency signals through. This is what the line is showing, there are only non-zero values for frequencies under 15Hz.

Q2: DataFrame indexing and slicing (20 Points)

1. Create a DataFrame with the same row indices, column labels, and data as the table below.

In [197…
```
data = {
    'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', '
    'age': [2.5, 3.0, 0.5, np.nan, 5.0, 2.0, 4.5, np.nan, 7.0, 3.0],
    'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']
}

index_labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

vet = pd.DataFrame(data, index=index_labels)

vet
```

Out[197…

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	2.0	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

2. Select only the 'animal' and 'visits' columns from the DataFrame and show the result.

In [197…
```
animal_visit = vet[["animal", "visits"]]
animal_visit
```

Out[197…

	animal	visits
a	cat	1
b	cat	3
c	snake	2
d	dog	3
e	dog	2
f	cat	3
g	snake	1
h	cat	1
i	dog	2
j	dog	1

3. Select the data in rows 'c', 'f', and 'g' and in columns 'animal' and 'age'. Show the result.

In [197…

```
animal_age = vet[["animal", "age"]]
animal_age_rows = animal_age.loc[['c', 'f', 'g']]
animal_age_rows
```

Out[197…

	animal	age
c	snake	0.5
f	cat	2.0
g	snake	4.5

4. Select the rows where the animal is a dog and the age is less than 5.

In [197…

```
young_dog = vet[(vet['animal'] == 'dog') & (vet['age'] < 5)]
young_dog
```

Out[197…

	animal	age	visits	priority
j	dog	3.0	1	no

5. In the 'animal' column, change the 'snake' entries to 'python'

In [197…

```
vet['animal'] = vet['animal'].replace('snake', 'python')
vet
```

Out[197...

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	python	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	2.0	3	no
g	python	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

6. Select the rows where the age is missing, i.e. age value is NaN.

In [198...

```
rows_with_nan_age = vet[vet['age'].isna()]
rows_with_nan_age
```

Out[198...

	animal	age	visits	priority
d	dog	NaN	3	yes
h	cat	NaN	1	yes

7. We want to clean up the NaN by replacing it with the average age of the dogs (or cats).

First, please find out the average age of the dogs and the average age of the cats.

In [198...

```
avg_dog_age = vet[vet['animal'] == 'dog']['age'].mean()
avg_cat_age = vet[vet['animal'] == 'cat']['age'].mean()
```

8. Replace the NaNs with the average ages and show the resulting DataFrame

In [198...

```
vet.loc[(vet['animal'] == 'dog') & (vet['age'].isna()), 'age'] = avg_dog_age
vet.loc[(vet['animal'] == 'cat') & (vet['age'].isna()), 'age'] = avg_cat_age
vet
```

Out[198...

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	python	0.5	2	no
d	dog	5.0	3	yes
e	dog	5.0	2	no
f	cat	2.0	3	no
g	python	4.5	1	no
h	cat	2.5	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

Q3: DataFrame Arithmetics (20 Points)

1. Generate a DataFrame that has 10 rows and 8 columns of random numbers (use np.random.rand).

Label the columns A, B, C, D, E, F, G and H.

In [198...

```
random_df = pd.DataFrame(np.random.rand(10, 8), columns=['A', 'B', 'C', 'D', 'E', '
random_df
```

Out[198...

	A	B	C	D	E	F	G	H
0	0.341590	0.286880	0.454877	0.002134	0.801347	0.214395	0.476336	0.672425
1	0.514033	0.913722	0.774748	0.661988	0.874113	0.680487	0.523161	0.534085
2	0.774247	0.618925	0.060184	0.188947	0.813340	0.074615	0.841043	0.296226
3	0.415018	0.607462	0.629101	0.593177	0.762782	0.611391	0.097456	0.614747
4	0.393269	0.903852	0.600840	0.546946	0.051540	0.862807	0.319565	0.039874
5	0.770410	0.086694	0.673915	0.900129	0.938677	0.974528	0.967040	0.037097
6	0.896251	0.963545	0.379707	0.320686	0.553815	0.969362	0.655121	0.419900
7	0.448668	0.730125	0.638986	0.131164	0.582692	0.155190	0.711499	0.302900
8	0.529794	0.778511	0.404318	0.080371	0.648186	0.387732	0.296235	0.084939
9	0.772952	0.045857	0.699523	0.160021	0.721055	0.111040	0.343675	0.989029

2. Calculate the sum of numbers for each column. Display the result.

In [198…
```
column_sums = random_df.sum()
column_sums
```

Out[198…
```
A    5.856233
B    5.935572
C    5.316199
D    3.585563
E    6.747548
F    5.041548
G    5.231131
H    3.991220
dtype: float64
```

3. Which column has the smallest sum? Please answer with the column label.

In [198…
```
col_min = column_sums.idxmin()
print("Column with the smallest sum:", col_min)
```

```
Column with the smallest sum: D
```

4. Calculate the mean value for each row, then subtract the mean from each element in the row.

In [198...
```
row_means = random_df.mean(axis=1)

df_centered = random_df.sub(row_means, axis=0)

df_centered
```

Out[198...

	A	B	C	D	E	F	G	H
0	-0.064658	-0.119368	0.048629	-0.404114	0.395099	-0.191853	0.070088	0.266177
1	-0.170509	0.229180	0.090206	-0.022554	0.189571	-0.004055	-0.161381	-0.150457
2	0.315806	0.160484	-0.398257	-0.269494	0.354900	-0.383826	0.382602	-0.162215
3	-0.126373	0.066070	0.087709	0.051786	0.221391	0.069999	-0.443936	0.073355
4	-0.071568	0.439015	0.136003	0.082110	-0.413296	0.397970	-0.145272	-0.424962
5	0.101849	-0.581868	0.005353	0.231568	0.270116	0.305967	0.298479	-0.631465
6	0.251453	0.318747	-0.265091	-0.324113	-0.090984	0.324563	0.010323	-0.224898
7	-0.013985	0.267472	0.176333	-0.331489	0.120039	-0.307463	0.248846	-0.159753
8	0.128533	0.377250	0.003058	-0.320890	0.246925	-0.013528	-0.105026	-0.316322
9	0.292558	-0.434537	0.219129	-0.320373	0.240661	-0.369354	-0.136719	0.508635

5. Add an additional row with the mean value of each column. Label this new row "Average".

In [198...
```
# Calculate the mean value for each column
column_means = random_df.mean()

# Append the column means as a new row labeled "Average"
random_df.loc['Average'] = column_means

random_df
```

Out[198...

	A	B	C	D	E	F	G	H
0	0.341590	0.286880	0.454877	0.002134	0.801347	0.214395	0.476336	0.672425
1	0.514033	0.913722	0.774748	0.661988	0.874113	0.680487	0.523161	0.534085
2	0.774247	0.618925	0.060184	0.188947	0.813340	0.074615	0.841043	0.296226
3	0.415018	0.607462	0.629101	0.593177	0.762782	0.611391	0.097456	0.614747
4	0.393269	0.903852	0.600840	0.546946	0.051540	0.862807	0.319565	0.039874
5	0.770410	0.086694	0.673915	0.900129	0.938677	0.974528	0.967040	0.037097
6	0.896251	0.963545	0.379707	0.320686	0.553815	0.969362	0.655121	0.419900
7	0.448668	0.730125	0.638986	0.131164	0.582692	0.155190	0.711499	0.302900
8	0.529794	0.778511	0.404318	0.080371	0.648186	0.387732	0.296235	0.084939
9	0.772952	0.045857	0.699523	0.160021	0.721055	0.111040	0.343675	0.989029
Average	0.585623	0.593557	0.531620	0.358556	0.674755	0.504155	0.523113	0.399122