

# lab3

December 8, 2024

CCE2203 Lab 3 - Lab report

Andy Debrincat 0135705L

Q.2.1

```
[55]: import os
      from scipy.io import wavfile
      import IPython.display as ipd

      fs, x = wavfile.read(os.path.join('data', 'imperfect_sos_cw_700_15_wpm.wav'))
      ipd.display(ipd.Audio(data=x, rate=fs))
```

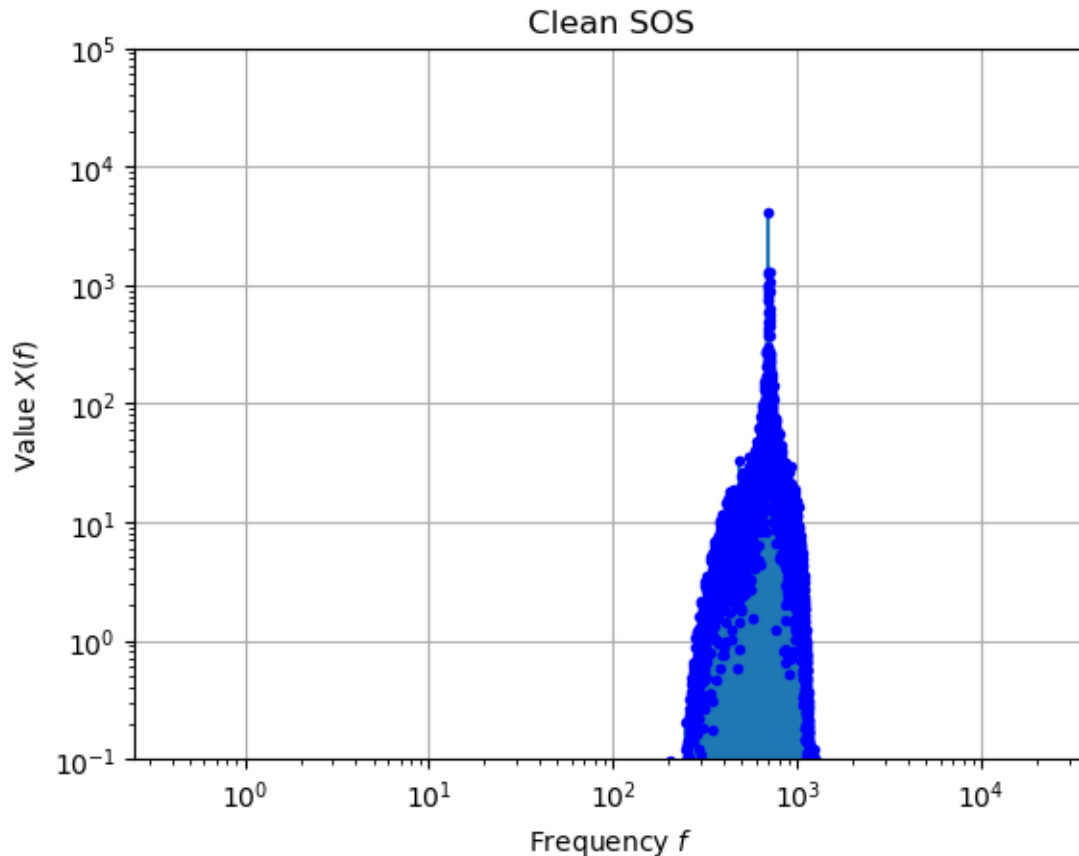
<IPython.lib.display.Audio object>

Q.2.2

```
[56]: import numpy as np
      X = np.fft.fft(x, norm='forward')
      F = np.fft.fftfreq(len(X), 1/fs)
```

Q.2.3

```
[57]: import matplotlib.pyplot as plt
      plt.figure()
      plt.stem(np.fft.fftshift(F), np.fft.fftshift(np.abs(X)), markerfmt='b.')
      plt.xscale('log')
      plt.yscale('log')
      plt.ylim(1e-1, 1e5)
      plt.grid(True)
      plt.xlabel('Frequency $f$')
      plt.ylabel('Value $X(f)$')
      plt.title('Clean SOS')
      plt.show()
```



#### Q.2.4

The frequencies present in the signal can be observed to be concentrated around 1000 Hz, while beyond this neighbourhood, the frequencies diminish to 0. The frequency of the carrier wave, can often be determined by looking at where the magnitude of the coefficient of the frequency is largest. This is because the carrier wave contributes the dominant frequency component in a modulated signal. Then we may calculate this using the following code:

```
[58]: max_index = np.argmax(np.abs(X))
      print(f"The maximum value of |X| occurs at frequency: {F[max_index]:.2f} Hz")
```

The maximum value of  $|X|$  occurs at frequency: 699.94 Hz

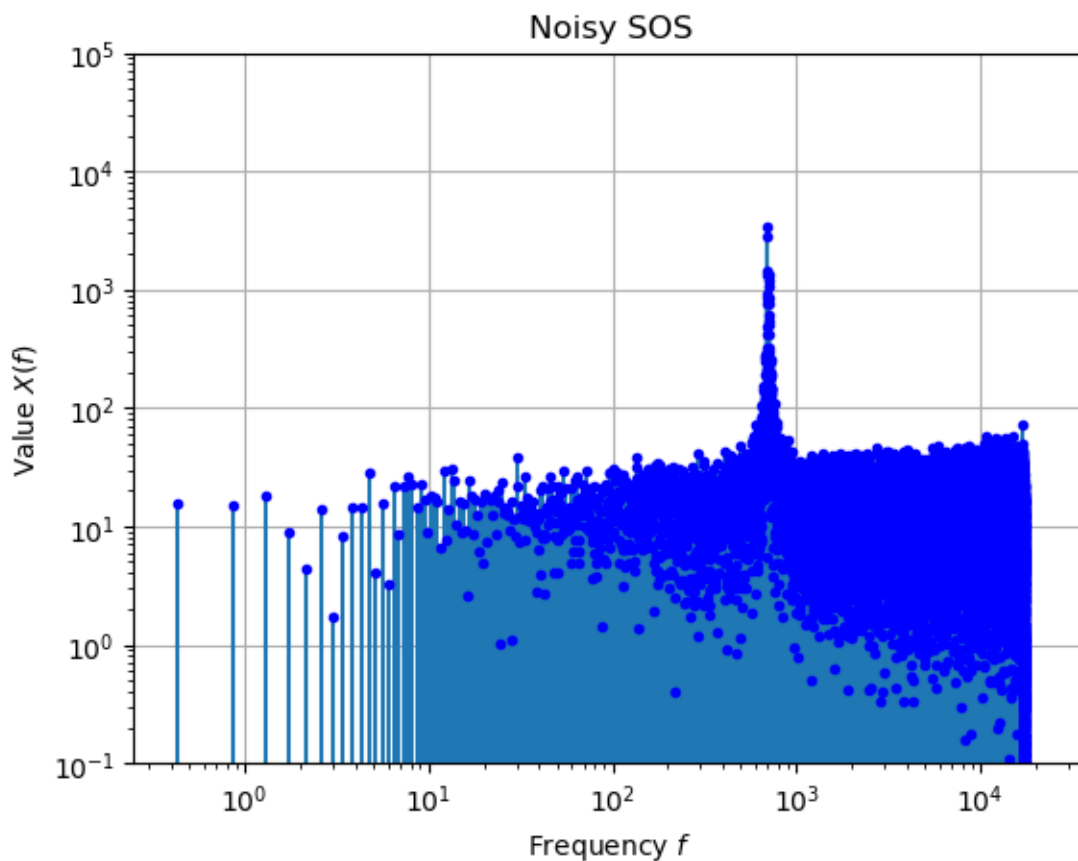
This means that most probably the carrier signal frequency is that of 699.94 Hz.

#### Q.2.5

```
[59]: fs2, x2 = wavfile.read(os.path.join('data', 'noisy_sos_cw_700_15_wpm.wav'))
      ipd.display(ipd.Audio(data=x2, rate=fs2))
```

<IPython.lib.display.Audio object>

```
[60]: X2 = np.fft.fft(x2,norm='forward')
F2 = np.fft.fftfreq(len(X2), 1/fs2)
plt.figure()
plt.stem(np.fft.fftshift(F2),np.fft.fftshift(np.abs(X2)),markerfmt='b. ')
plt.xscale('log')
plt.yscale('log')
plt.ylim(1e-1,1e5)
plt.grid(True)
plt.xlabel('Frequency $f$')
plt.ylabel('Value $X(f)$')
plt.title('Noisy SOS')
plt.show()
```



### Q.2.6

Unlike the clean SOS signal, the frequency coefficients are no longer all concentrated around the same range as before. There is now a high concentration in the higher frequencies, where previously there was none.

We also notice a few components in the lower frequencies. However, since the graph is on a logarithmic scale and these components are relatively few, they do not contribute significantly to

the noise. The primary difference contributing to most of the noise is the high concentration of energy in the higher frequencies.

This may imply that noise tends to lie in the high frequency range.

It can also be noted that the frequency components of the clean SOS seem to be amplified, and have greater values than before. This could be due to the noise having some overlapping frequencies which superimpose their magnitudes onto the clean SOS's frequency components.

### Q.3.1

From the clean SOS, it can be observed that all the frequencies with coefficients of non-negligible magnitude (greater than  $10^{-1}$ ) are concentrated in a band around  $10^3$  Hz. To determine the exact range of this band, the following snippet was used:

```
[61]: threshold = 0.1

positive_indices = F >= 0
F_positive = F[positive_indices]
X_positive = np.abs(X[positive_indices])

indices = np.where(X_positive > threshold)[0]

first_frequency = F_positive[indices[0]]
last_frequency = F_positive[indices[-1]]
print(f"The first positive frequency with |X| > {threshold} is {first_frequency:.2f} Hz")
print(f"The last positive frequency with |X| > {threshold} is {last_frequency:.2f} Hz")
```

The first positive frequency with  $|X| > 0.1$  is 248.39 Hz

The last positive frequency with  $|X| > 0.1$  is 1238.92 Hz

Rounding to the nearest 50, it can be concluded that the range [250, 1250] is of most interest. Anything else not in that frequency range, is not present in the original signal and is most likely noise.

### Q.3.2

To create a filter that will zero out all frequencies beyond the range [250, 1250], we may use the fact that convolution in the time domain corresponds to multiplication in the frequency domain, in this way, we can simply multiply the array 'X2' (noisy signal), by an array of the exact same size and length filled with zeros everywhere except in the desired range. To cater for the fact that we also have negative frequencies, which mirror the positive frequencies, we also have to add ones in the range [-1250, -250]. This can be done through the following code snippet:

```
[62]: Filter = np.zeros_like(X2)
for i in range(len(Filter) - 1, -1, -1):
    if((250 <= F2[i] <= 1250) or (-1250 <= F2[i] <= -250)):
```

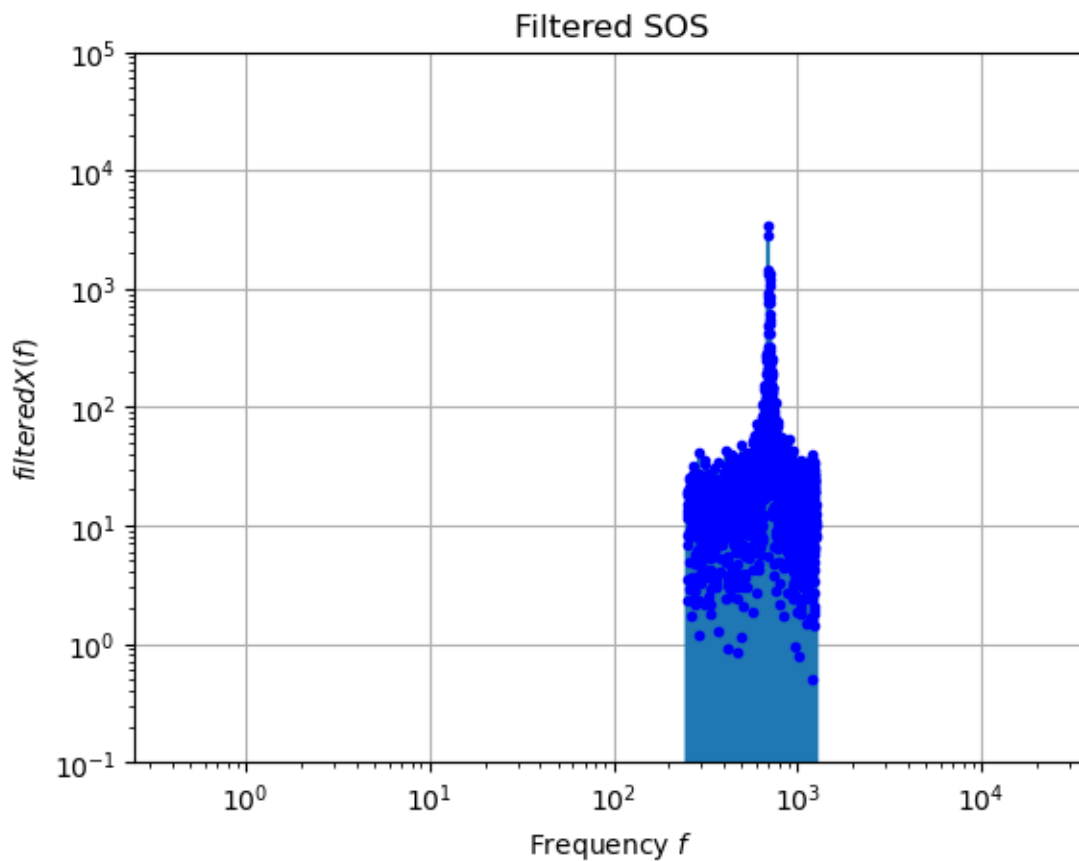
```

        Filter[i] = 1
    else:
        Filter[i] = 0

Filtered_freq_response = X2*Filter

plt.figure()
plt.stem(np.fft.fftshift(F2),np.fft.fftshift(np.
    ↪abs(Filtered_freq_response)),markerfmt='b. ')
plt.xscale('log')
plt.yscale('log')
plt.ylim(1e-1,1e5)
plt.grid(True)
plt.xlabel('Frequency $f$')
plt.ylabel('$filtered X(f)$')
plt.title('Filtered SOS')
plt.show()

```

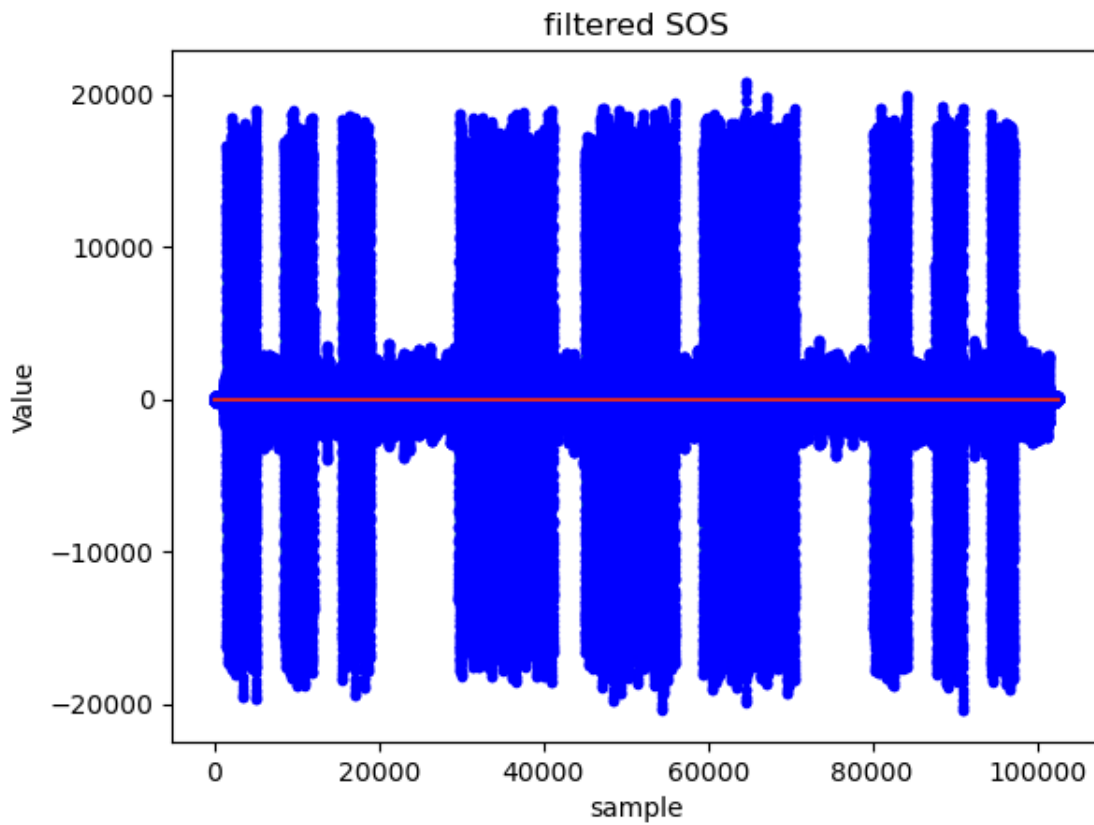


Q.3.3

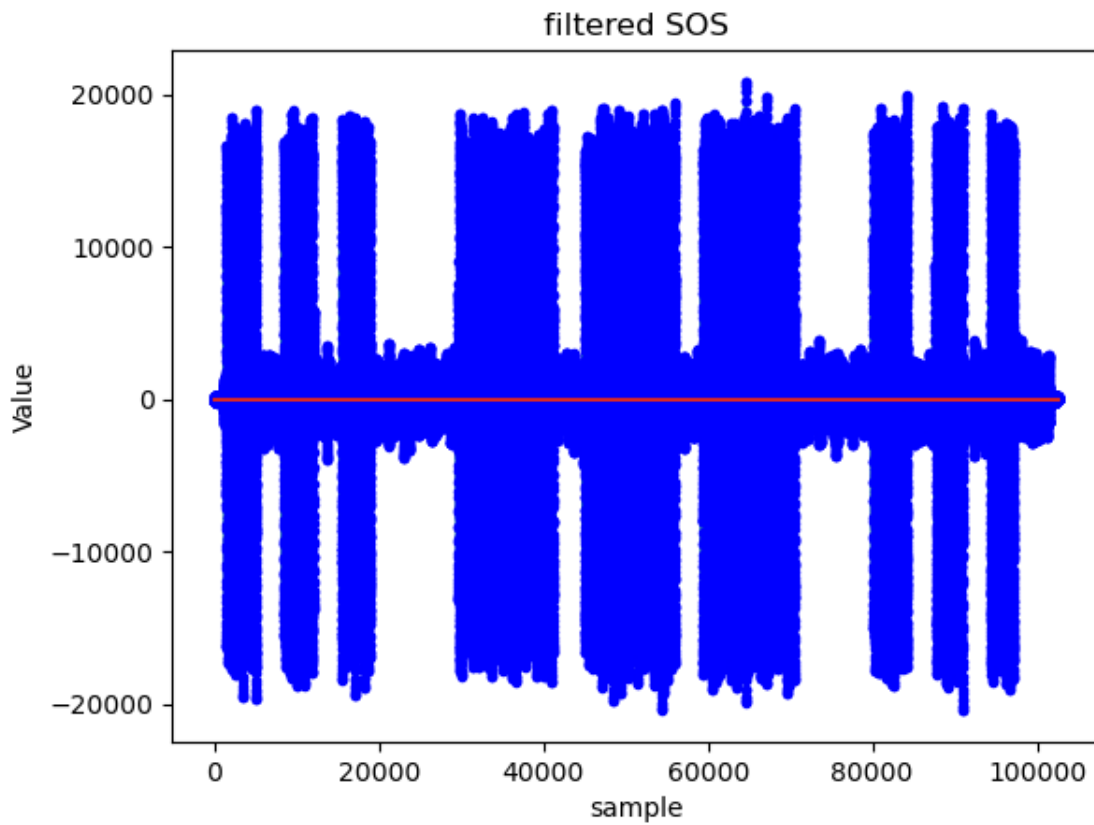
```
[63]: filtered_x = np.fft.ifft(Filtered_freq_response, norm = "forward")
      filtered_x = filtered_x.real
```

Q.3.4

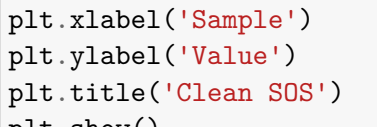
```
[64]: plt.figure()
      plt.stem(filtered_x,markerfmt='b.')
```



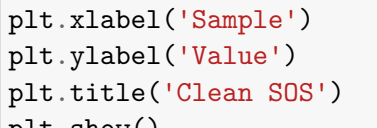
```
      plt.xlabel('sample')
      plt.ylabel('Value')
      plt.title('filtered SOS')
      plt.show()
```

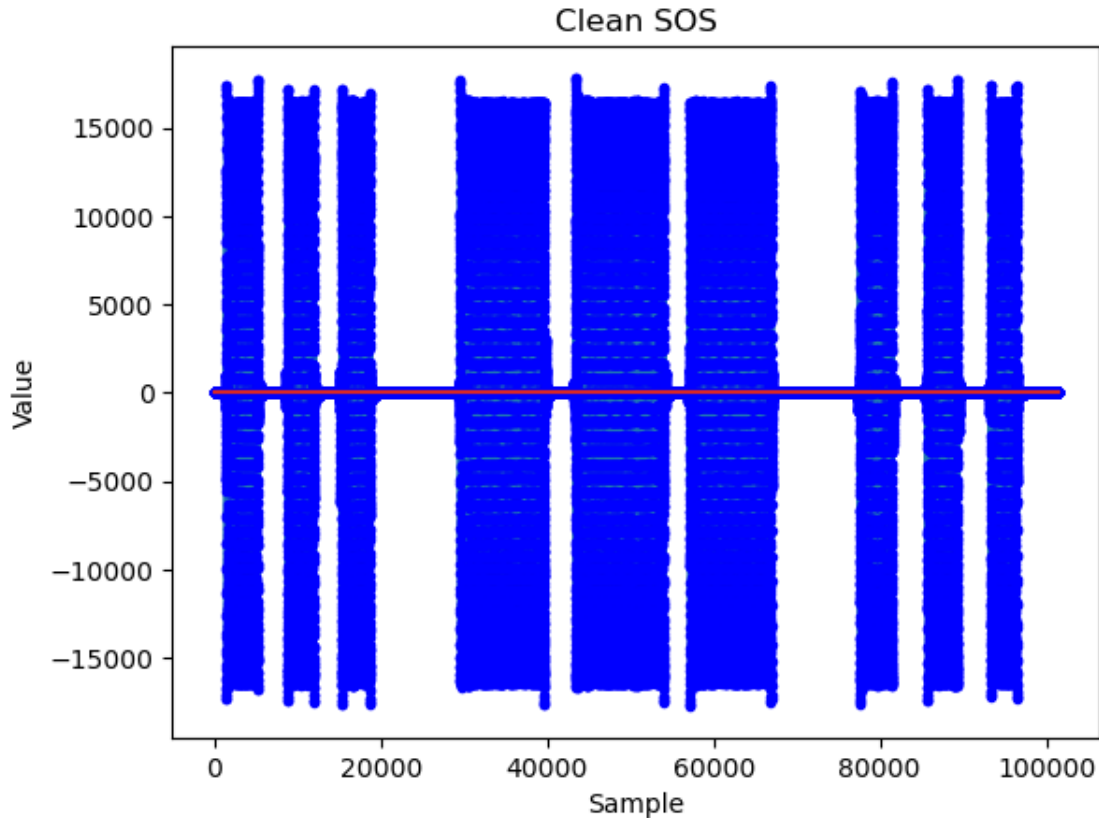


```
[65]: plt.figure()
      plt.stem(x,markerfmt='b.')
```



```
      plt.xlabel('Sample')
      plt.ylabel('Value')
      plt.title('Clean SOS')
      plt.show()
```





The result obtained after removing the coefficients of the frequencies not present in the original signal is much cleaner than the noisy signal and very close to the clean SOS. However, it can be observed that the amplitude of the signal increased by approximately 2000, and there is low-amplitude wave where there should be silence. The reasons for these errors could be that the noise in the frequency range [250, 1250] was left unaltered and no effort was made to remove it. In addition, the range was rounded and some desired frequencies could have been removed.

Q.3.5

```
[66]: ipd.display(ipd.Audio(data=filtered_x, rate=fs2))
```

<IPython.lib.display.Audio object>

In comparison to the noisy signal, this sounds much cleaner. However, the low amplitude noise mentioned previously can clearly be heard in the background.

This approach was quite effective in removing the unwanted noise. Considering how intuitive and straightforward the process is, it offers a quick and simple method to significantly improve a signal when noise is an issue.

This method could be used before applying another filter to remove the noise within the frequency range itself, to make the process easier.

The low amplitude noise could easily be removed if it is an issue, by simply multiplying by 0 when

there should be silence.