lab1

November 5, 2024

Andy Debrincat 0135705L

Signals ans Systems CCE2203

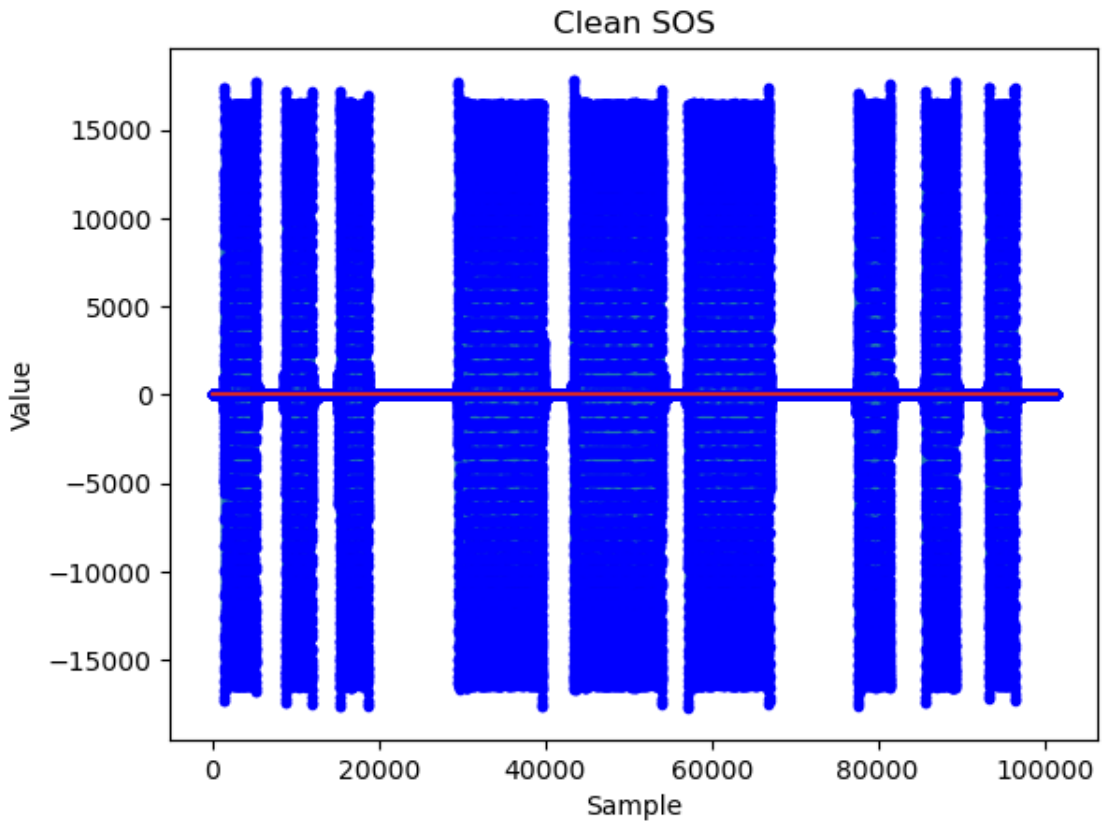
Q 2.1

[16]:
```
import os
from scipy.io import wavfile
fs, sos_clean = wavfile.read(os.path.join('data','imperfect_sos_cw_700_15_wpm.
 ↪wav'))
import IPython.display as ipd
ipd.display(ipd.Audio(data=sos_clean, rate=fs))
```

```
<IPython.lib.display.Audio object>
```

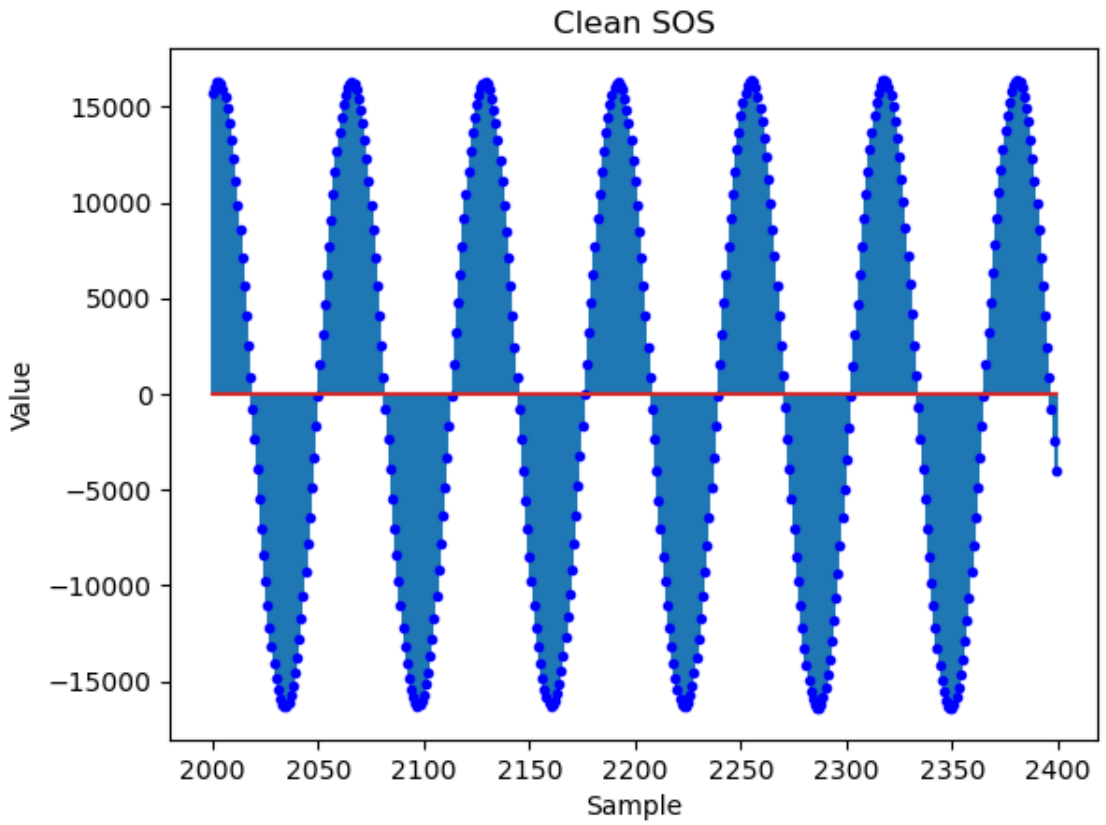
Q 2.2

[17]:
```
import matplotlib.pyplot as plt
plt.figure()
plt.stem(sos_clean,markerfmt='b.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Clean SOS')
plt.show()
```



Q 2.3

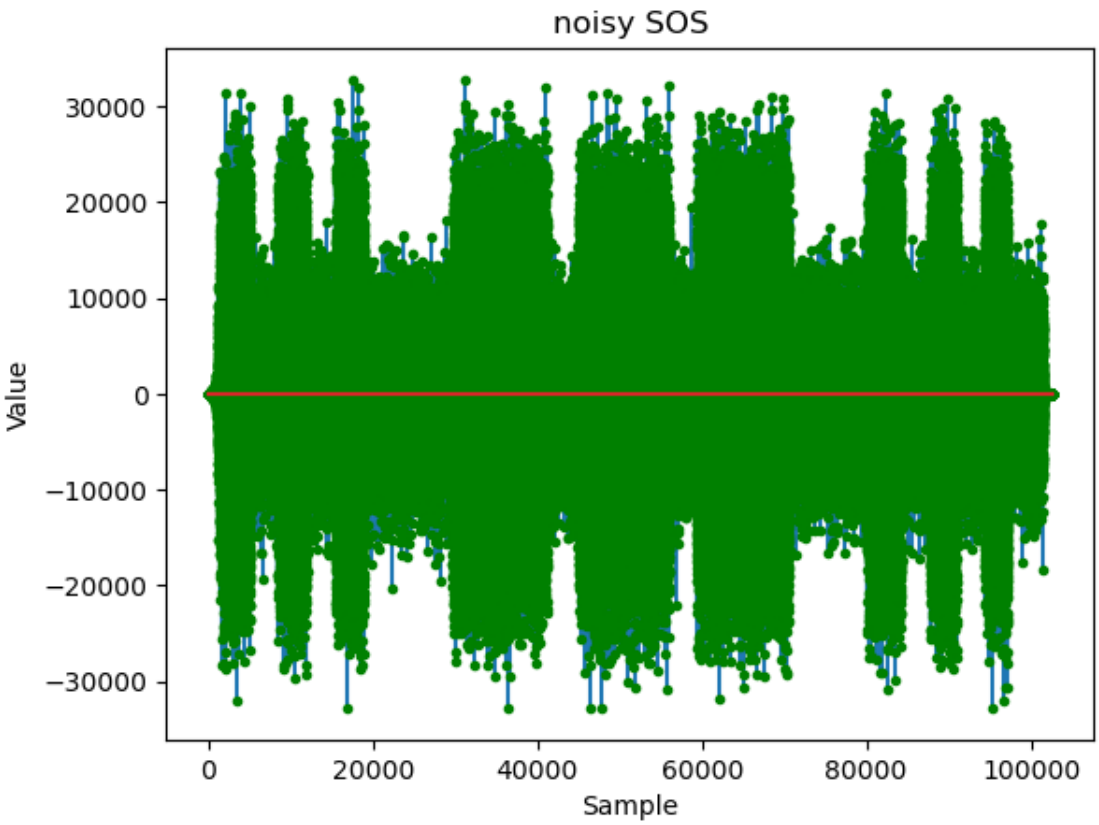
[ ]:
```
plt.figure()
t = range(2000,2400)
plt.stem(t,sos_clean[t],markerfmt='b.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Clean SOS(zoomed in)')
plt.show()
```

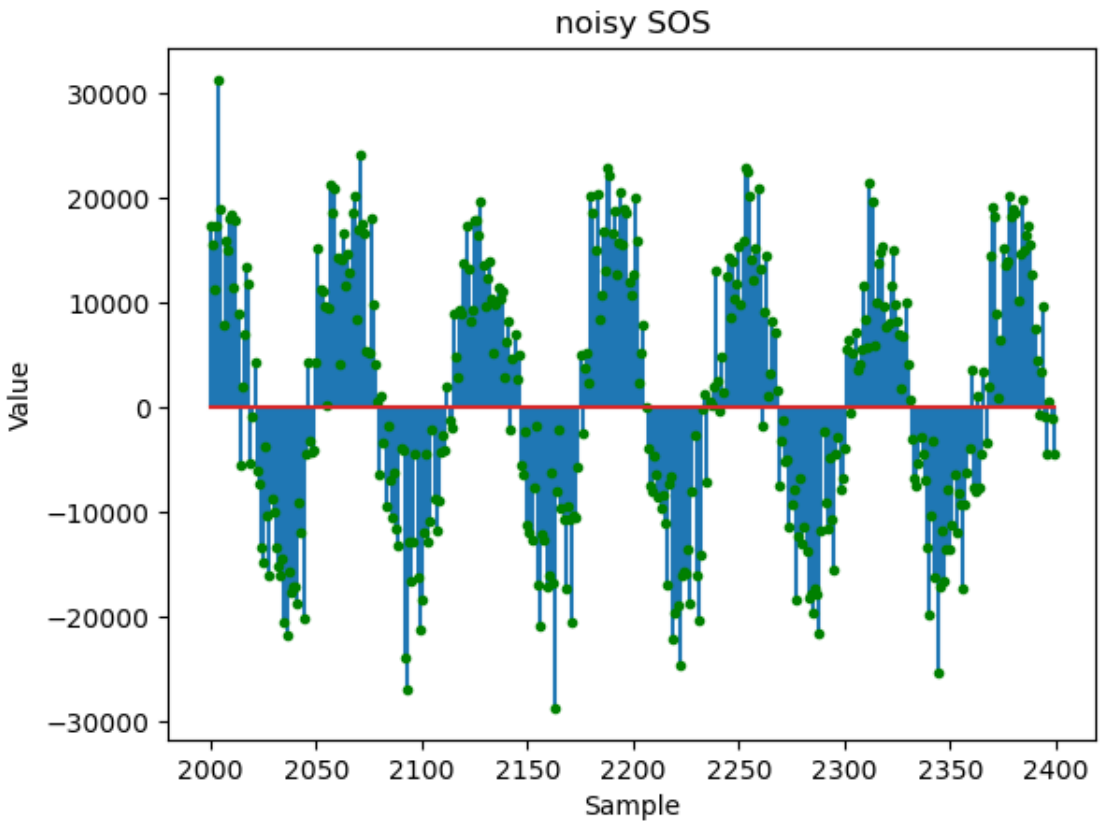


Q 3.1

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

plt.figure()
plt.stem(sos_noisy,markerfmt='g.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('noisy SOS')
plt.show()

plt.figure()
t = range(2000,2400)
plt.stem(t,sos_noisy[t],markerfmt='g.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('noisy SOS(zoomed in)')
plt.show()
```

```
<IPython.lib.display.Audio object>
```


noisy SOS
30000
20000
10000
0
−10000
−20000
−30000
Value
0
20000
40000
60000
80000
100000
Sample



Q 3.2

In the first question it can be seen that the signal when a beep is supposed to be represented is simply f(n) = Asin(s.n) where A is approximately 15000 and s.n is simply the discrete increment at which we are varying the input. In this case, 2 seconds of audio gave about 100000 samples, meaning there was about 50000 samples per second, so if the input n is time in seconds, s is about 1/50000.

Then when there is supposed to be a pause signifying the pause in morse code, there is complete silence and the function is f(n) = 0.

(This is simply an approximation so the values obtained are not accurate)

The signal sounds clean and the message is clearly that of SOS.

In the second question when hearing the signal we may immediately notice that there is some other noise in the signal that should not be there. When looking at the zoomed out graph, it can be seen that the signal has ragged edges when it is supposed to be conveying the beeps in morse code, and is no longer 0 when it is supposed to be conveying silence. We may also notice that the bursts now sometimes reach 20000 and above and are no longer clearly bounded by approximately 15000.

When zooming in, it is clear that the near perfect sine wave sampled approximately 50000 times per second, is now a very rough and poor imitation of one. However the general shape is still there and it still looks like it is trying to copy a sin wave.

The signal sounds heavily distorted and loud, however the beeps can be distinguished.

Q 4.1

Here a filter that will average out a function in a window of size 5, will be of the form h(n) = 1/5(delta(s(n-2)) + delta(s.(n-1)) + delta(s.n) + delta(s.(n+1)) + delta(s.(n+2))). In general for a window average of size k, we simply need to adjust h(n) to be multiplied by 1/k instead of 1/5, and add terms in delta from (s.(n - floor(k/2)))to (s.(n + floor(k/2))) when k is odd, and when k is even the range should be from (s.(k/2))to (s.(k/2 -1)). In this way the window will be of length exactly k. And h will have value 1/k for all these k values of n. So when the window is dragged along x(n), it will take exactly k consecutive values of x(n) and multiply each by 1/k and add them. Effectively taking the moving average in a window of length k.

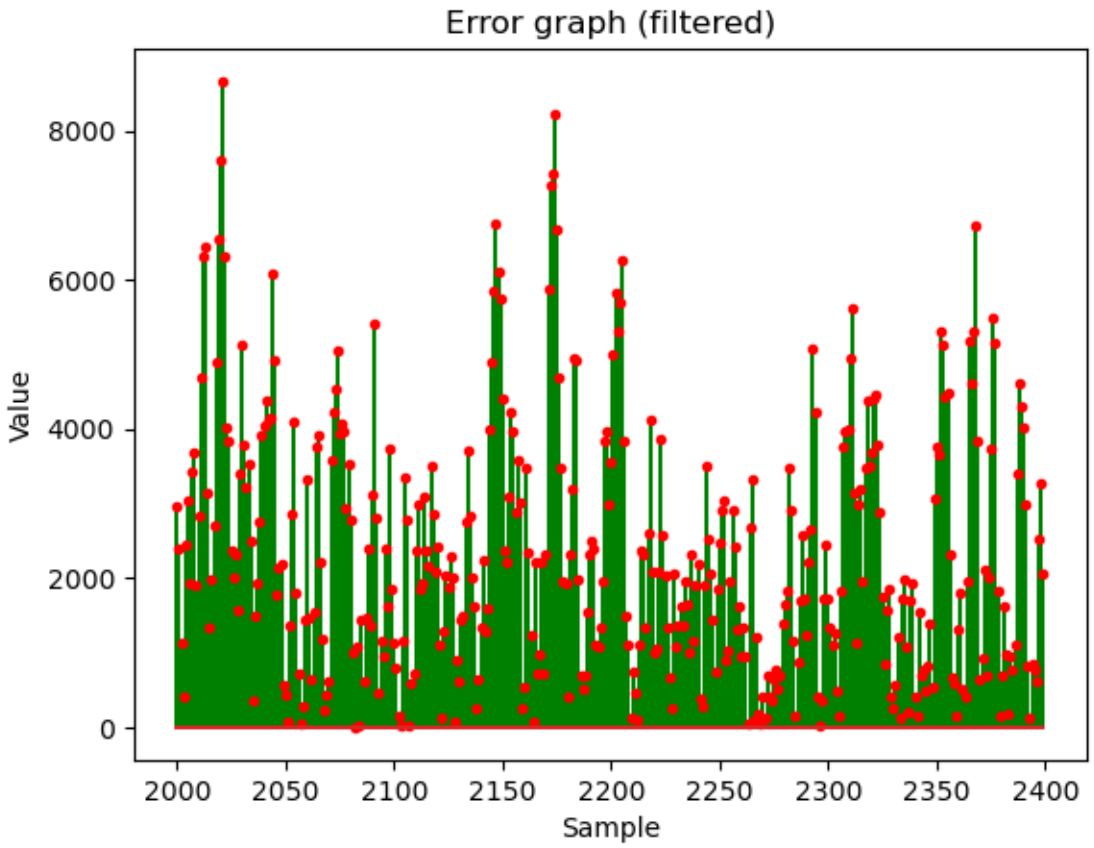
[20]:
```
n = 5
h = [1 / n for i in range(n)]
```

Q 4.2

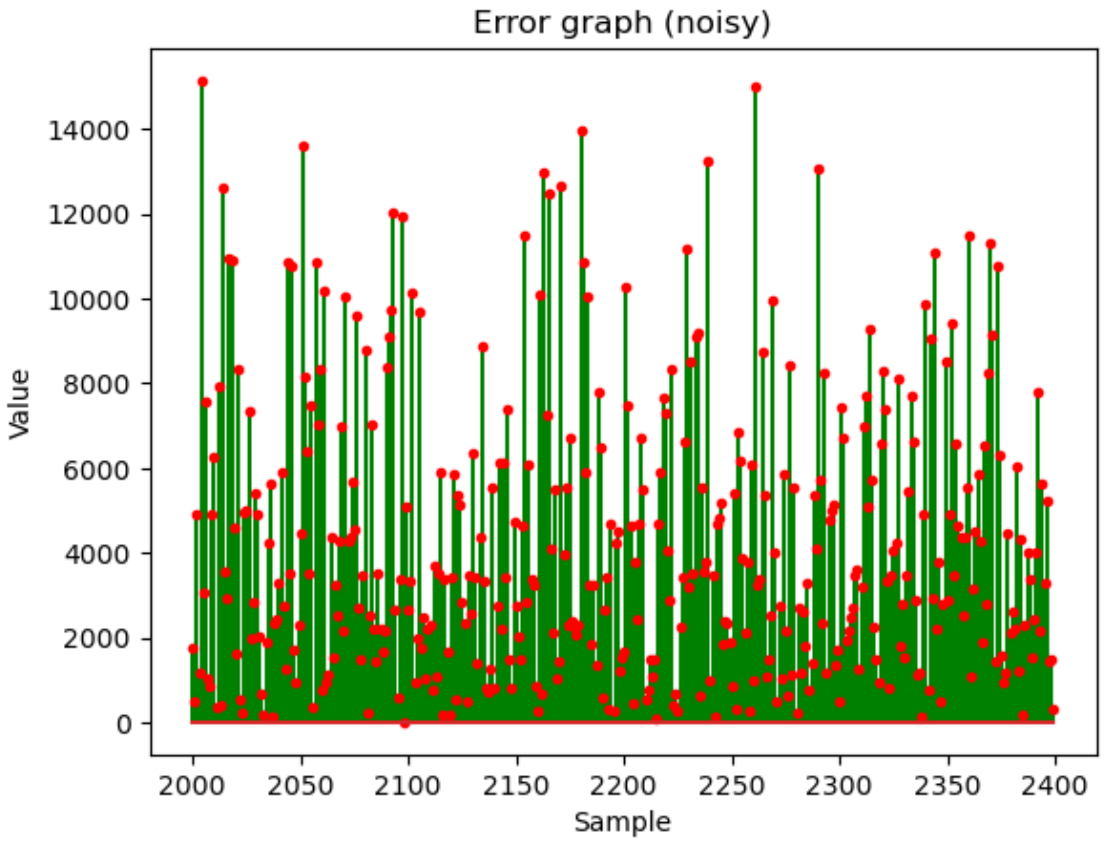
[21]:
```
from scipy import signal
sos_filtered = signal.convolve(sos_noisy,h)
```

Q 4.3

[51]:
```
temp = abs(sos_clean[t] - sos_filtered[t])
plt.figure()
plt.stem(t,temp,linefmt ='g' ,markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Error graph (filtered)')
plt.show()

import numpy as np
average = np.mean(temp)
print("The average error is : " , average)



temp = abs(sos_clean[t] - sos_noisy[t])
plt.figure()
plt.stem(t,temp,linefmt ='g' ,markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Error graph (noisy)')
plt.show()

average = np.mean(temp)
print("The average error is : " , average)
```




```
The average error is :  2305.3615
```




```
The average error is :  4160.4125
```

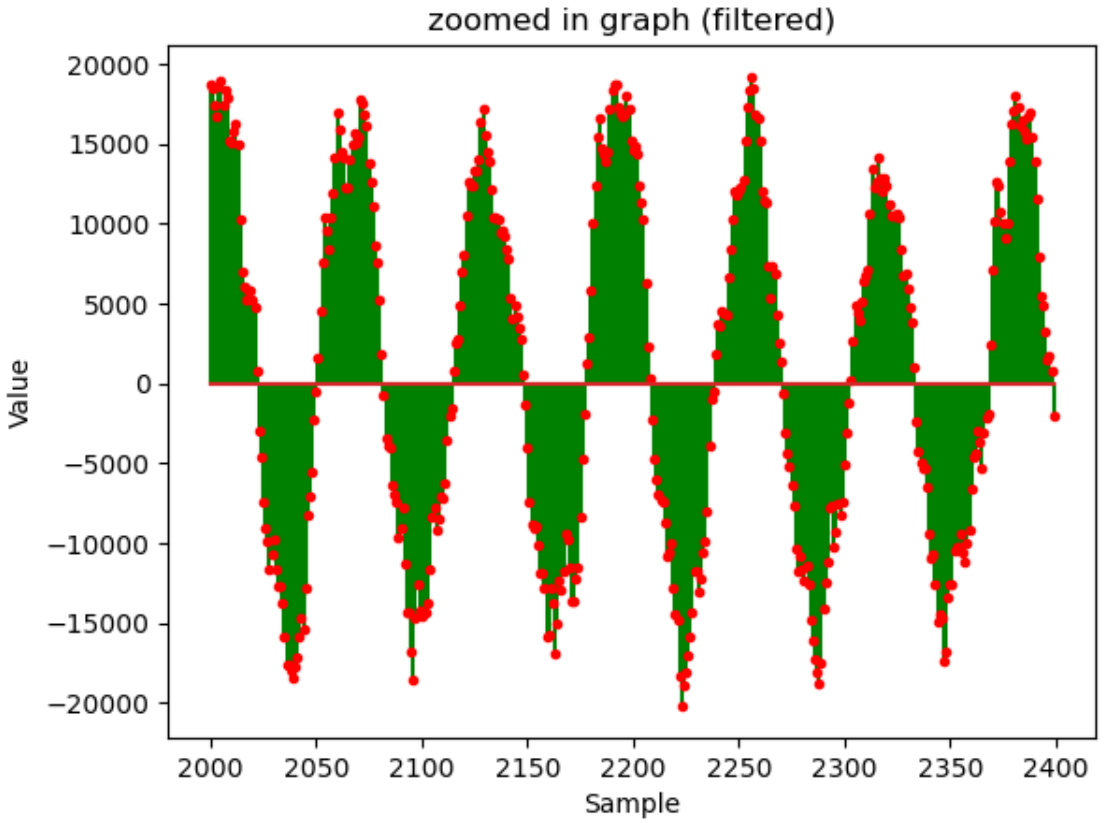
To compare the noisy signal and the filtered signal, the absolute value of the difference at each point of the filtered signal to each point of the original clean SOS signal was plotted. This was done also for the noisy signal. Note that a zoomed in sample was taken to better understand what is going on, because if this was done with the whole sample the errors would not make sense. Note also that this could only be done because the filter left the sine waves approximately in phase, meaning it did not shift or stretch the sine waves.

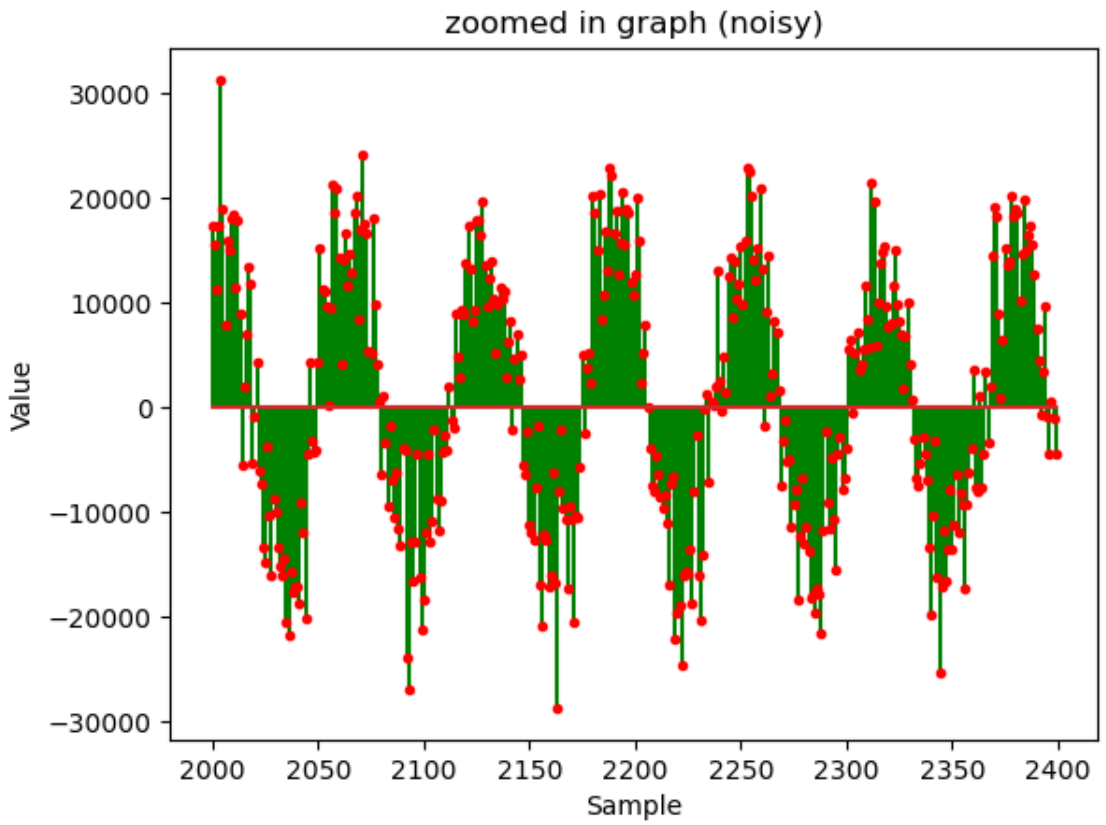
It can be seen that the errors in the filtered graph are at maximum 8000 (approximately), while the noisy graph has errors with a maximum of 14000(approximately).

In addition to this, the average error in the filtered graph is around 2000 while the average error in the noisy graph is about 4000.

[52]:
```
plt.figure()
plt.stem(t,sos_filtered[t],linefmt ='g' ,markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('zoomed in graph (filtered)')
plt.show()
```

```
plt.figure()
plt.stem(t,sos_noisy[t],linefmt ='g' ,markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('zoomed in graph (noisy)')
plt.show()
```





When comparing the zoomed in graphs, we may see that the filtered graph better approximates a sine wave than the noisy graph, although still very erroneous.

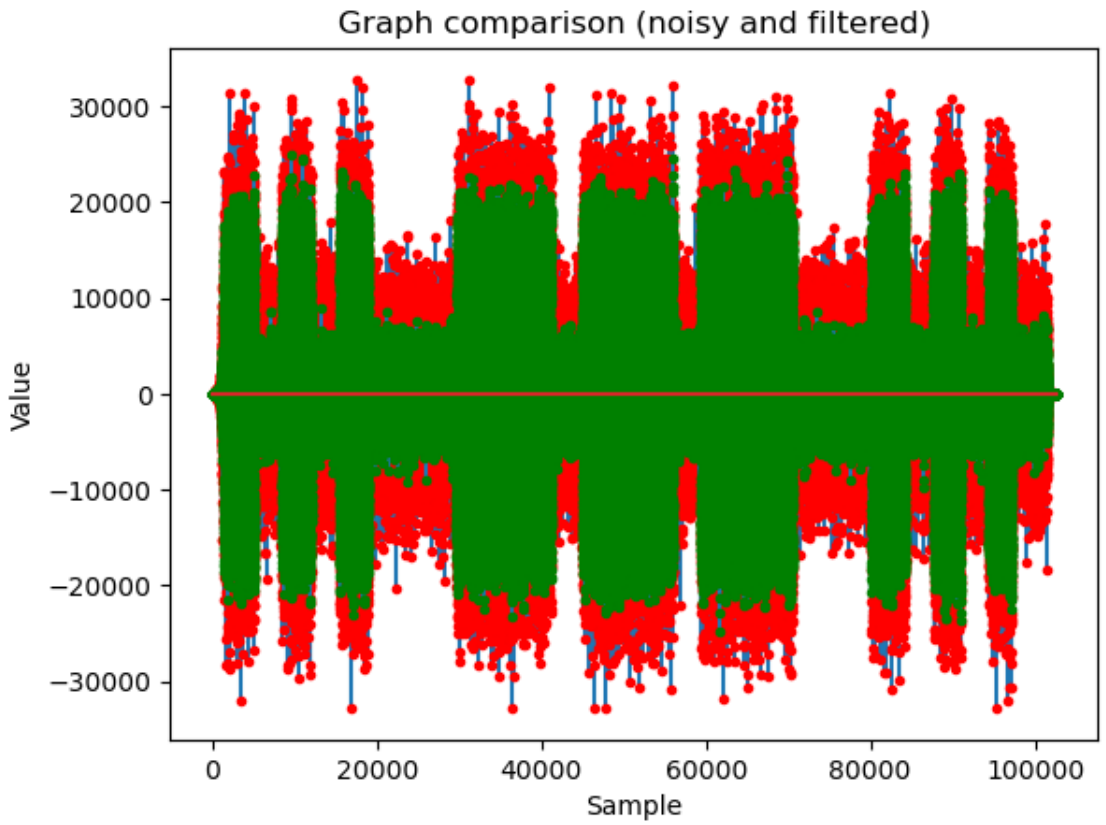
[53]:
```
ipd.display(ipd.Audio(data=sos_filtered, rate=fs))
ipd.display(ipd.Audio(data=sos_noisy, rate=fs))
```

```
<IPython.lib.display.Audio object>
```

```
<IPython.lib.display.Audio object>
```

Moreover when listening to the signal, the static in the noisy signal is much louder than the static in the filtered signal.

[ ]:
```
plt.figure()
plt.stem(sos_noisy,markerfmt='r.')
plt.stem(sos_filtered,markerfmt='g.')

plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Graph comparison (noisy and filtered)')
plt.show()
```



Another way in which the signals can be compared is to lay them out on top of eachother.

In the diagram the green graph is the filtered signal, while the red one is the noisy signal. It can be seen that when there is supposed to be silence, the filtered graph has an amplitude of approximately 8000 while the red graph has approximately 15000.

In addition to this the clean SOS has amplitude approximately 15000 when a line in morse code is supposed to be conveyed; in the graph it can be seen that the filtered graph has amplitude of approximately 20000 and the noisy graph has amplitude of approximately 25000.

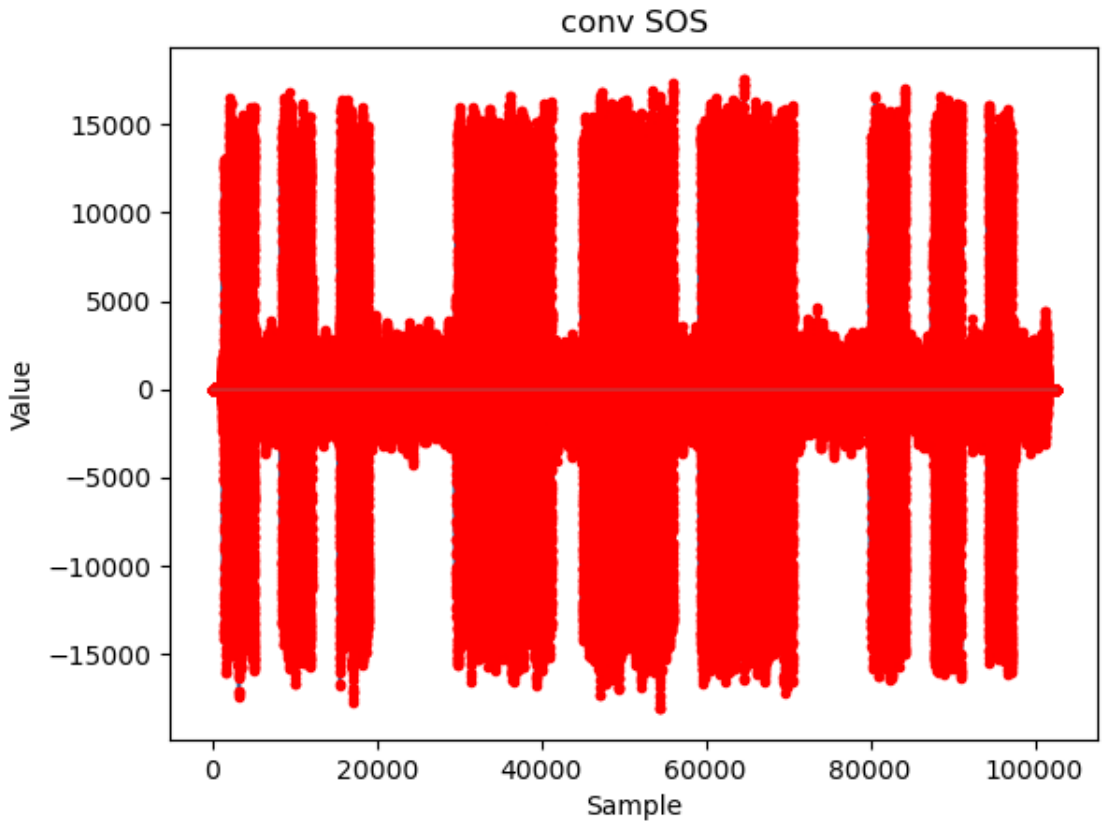
All of the above points towards the fact that the filtered signal is closer to the clean SOS signal than the noisy signal, although still not perfect.

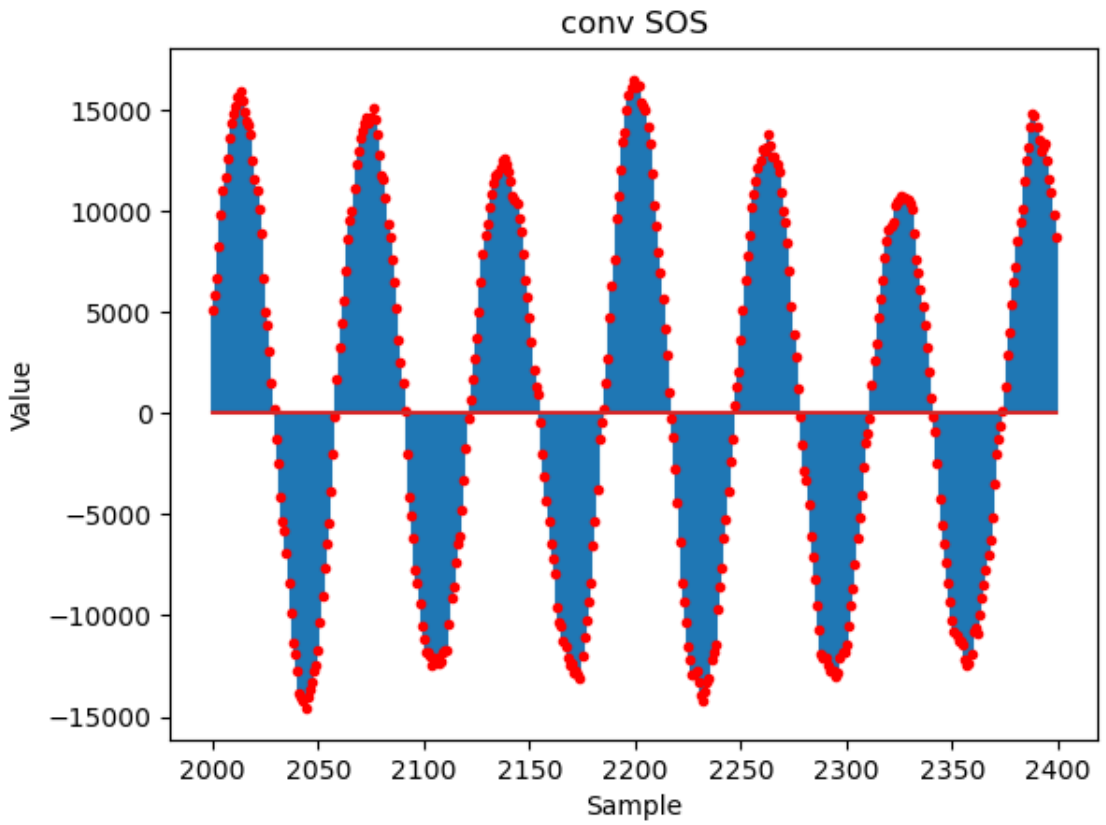
Q 4.4

The value of n has been increased to 20. This value was chosen after some testing for a more notable result.

[ ]:
```
n = 20
h = [1 / n for i in range(n)]
sos_filtered_2 = signal.convolve(sos_noisy,h)
plt.figure()
plt.stem(sos_filtered_2,markerfmt='r.')
```

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

plt.figure()
plt.stem(t,sos_filtered_2[t],markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('filtered SOS(zoomed in)')
plt.show()
ipd.display(ipd.Audio(data=sos_filtered_2, rate=fs))
```






```
<IPython.lib.display.Audio object>
```

Now that n has been increased slightly to 20, we can see that there are some improvements to the case when n was 5.

The amplitude is now approximately 15000 like the clean SOS, and when the signal is supposed to convey silence, the filtered signal has amplitude of approximately 3000.

When zooming in, it can be seen that the graph is much closer to a sine wave than the previous case, although not perfect.

When listening to the audio, the static noise sounds quieter than the case when n = 5.

All this shows that the filtered signal is an improvement over the case of n = 5.

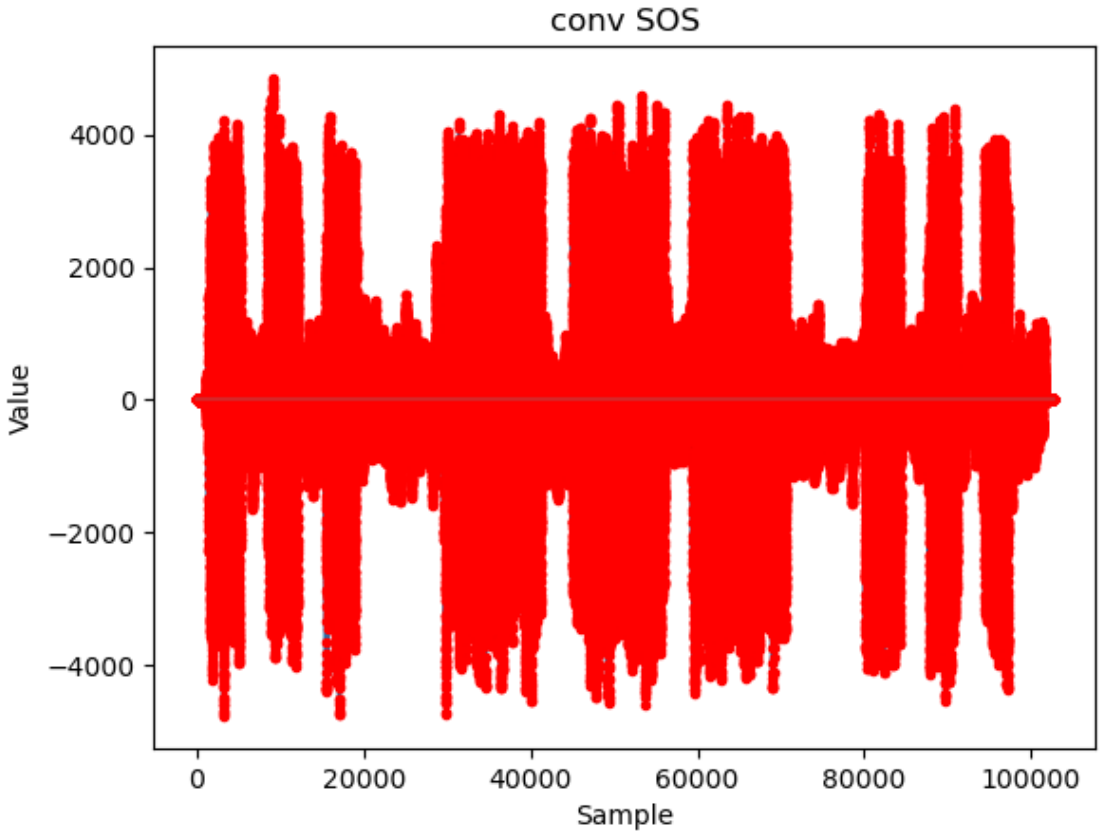
However noise is still present and the signal still remains imperfect.

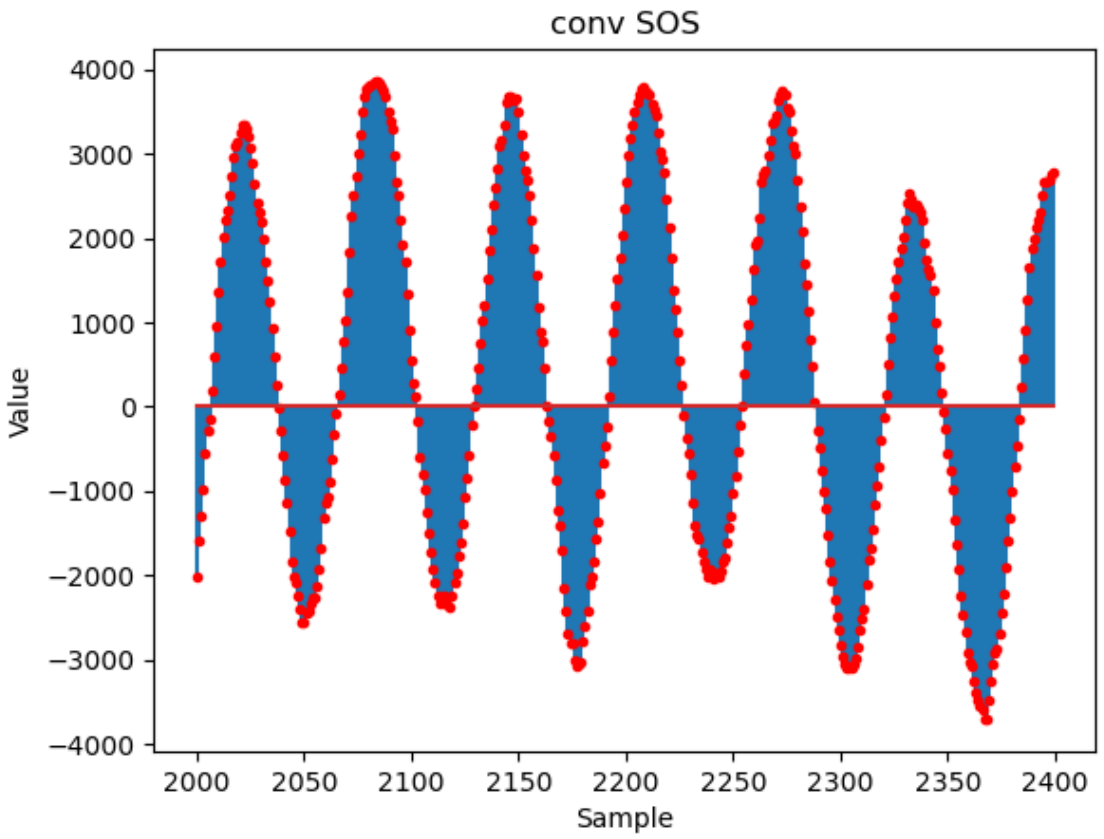
Q 4.5

The value of n will now be varied to 100 then to 1000.

[ ]:
```
n = 100
h = [1 / n for i in range(n)]
sos_filtered_3 = signal.convolve(sos_noisy,h)
plt.figure()
```

```
plt.stem(sos_filtered_3,markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('filtered SOS')
plt.show()
plt.figure()
plt.stem(t,sos_filtered_3[t],markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('filtered SOS(zoomed in)')
plt.show()
ipd.display(ipd.Audio(data=sos_filtered_3, rate=fs))
```






```
<IPython.lib.display.Audio object>
```

When n = 100, we can see that the signal's amplitude has been reduced to approximately 4000 when a beep in morse code is to be conveyed and approximately 1000 when there is supposed to be silence.

The zoomed in graph approximates a sine wave in the same way that the case for n = 20 did, however the amplitude is now lower.

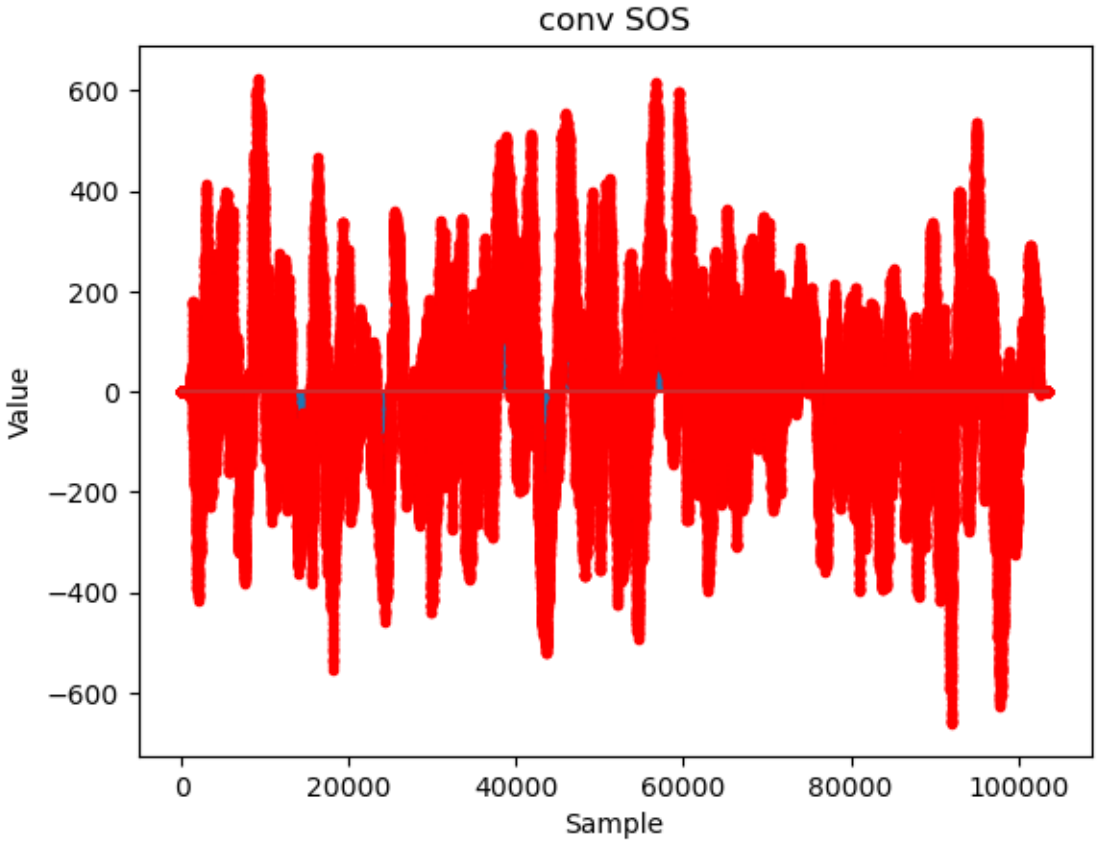
The audio sounds quieter than before.

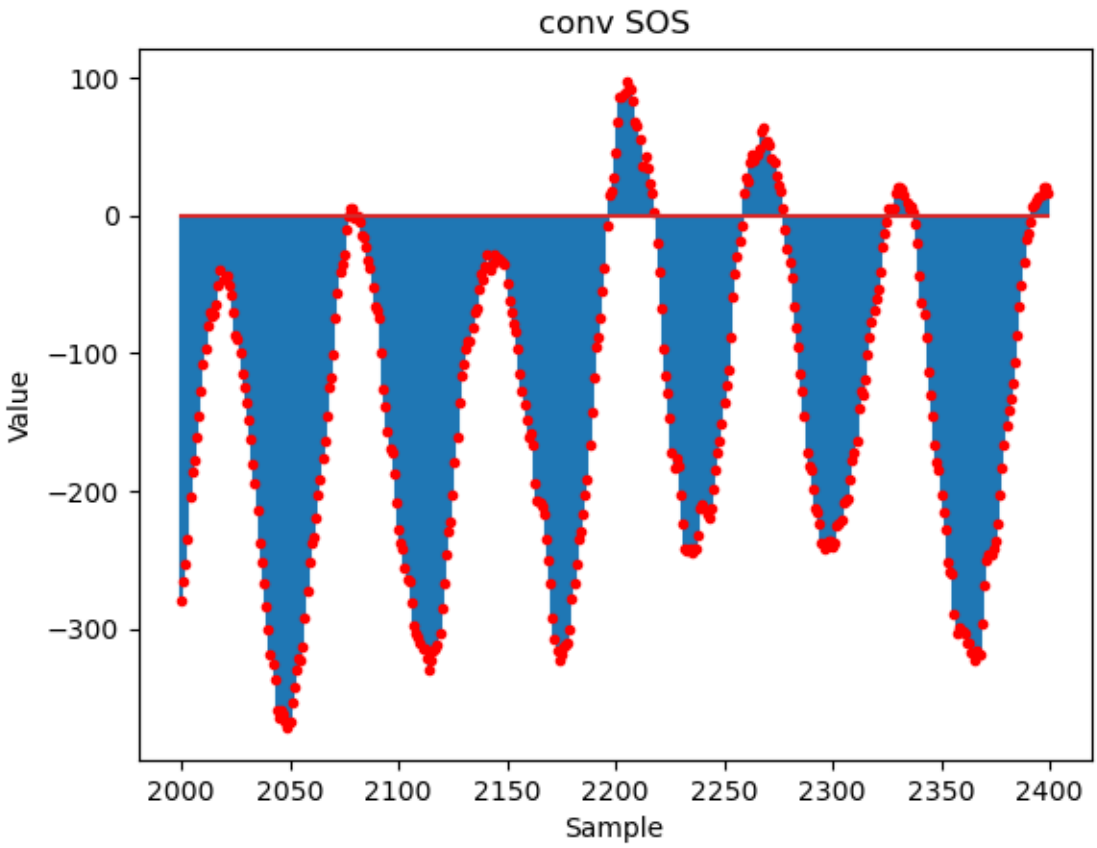
This shows that although the noise has decreased, it is still relatively large since now the whole signal has been dampened to an amplitude of 4000. This dampening effect may not always be desired if the exact shape of the clean SOS signal is to be maintained.

case when n = 1000

[ ]:
```
n = 1000
h = [1 / n for i in range(n)]
sos_filtered_4 = signal.convolve(sos_noisy,h)
plt.figure()
plt.stem(sos_filtered_4,markerfmt='r.')
plt.xlabel('Sample')
```

```
plt.ylabel('Value')
plt.title('filtered SOS')
plt.show()
plt.figure()
plt.stem(t,sos_filtered_4[t],markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('filtered SOS')
plt.show()
ipd.display(ipd.Audio(data=sos_filtered_4, rate=fs))
```






```
<IPython.lib.display.Audio object>
```

For the case when n = 1000, it can be seen the signal is now heavily dampened to an amplitude of approximately 600. It can also be seen that the graph is now distorted where one can barely distinguish where the morse code lines and where the silence should be. When zooming in, the graph does not approximate a sine wave at all. The audio signal still can be identified as an SOS message, however it sounds quiet and the noise sounds relatively louder.

All of this shows that increasing n greatly will distort the signal.

It can be observed that there is some limit to how much the value of n can be increased, in this case it seems that when n = 20, the convolved signal had the best result.

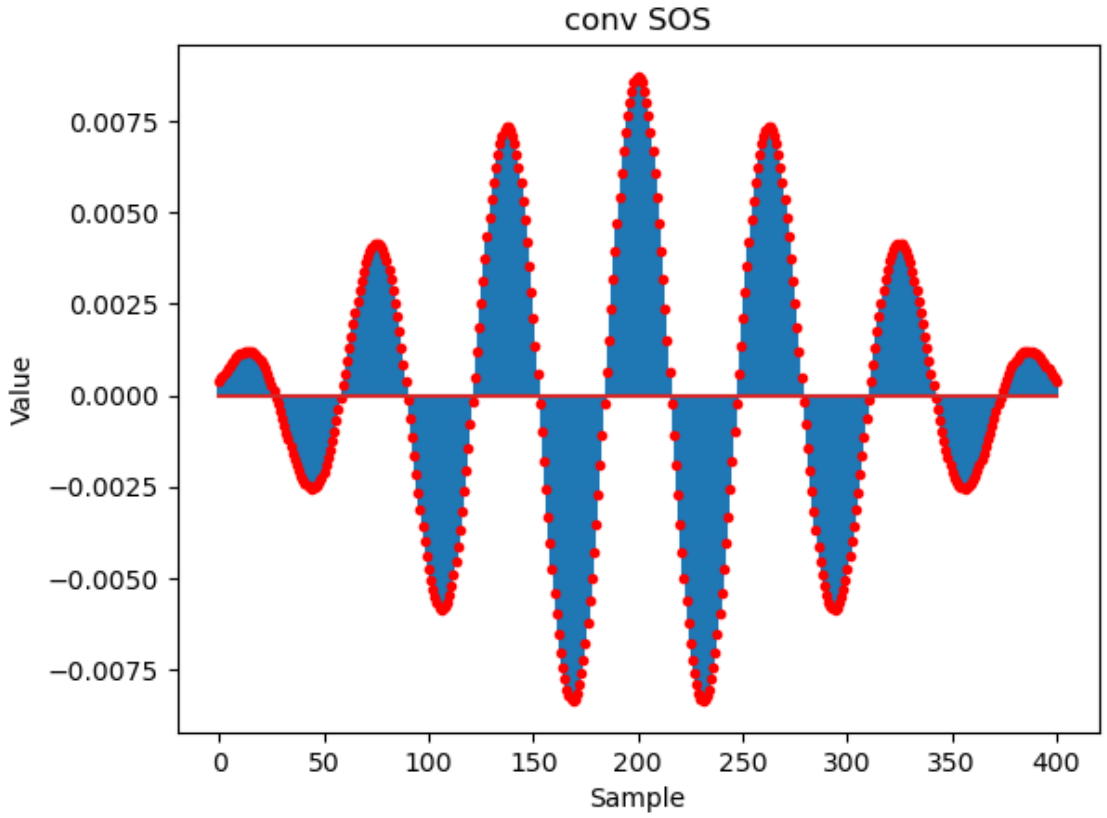
Q 5.1

[ ]:
```
import numpy as np
band_pass = np.load(os.path.join('data','filter.npy'))
```

Q 5.2

[ ]:
```
plt.figure()
plt.stem(band_pass,markerfmt='r.')
plt.xlabel('Sample')
```

```
plt.ylabel('Value')
plt.title('conv SOS')
plt.show()
plt.figure()
```

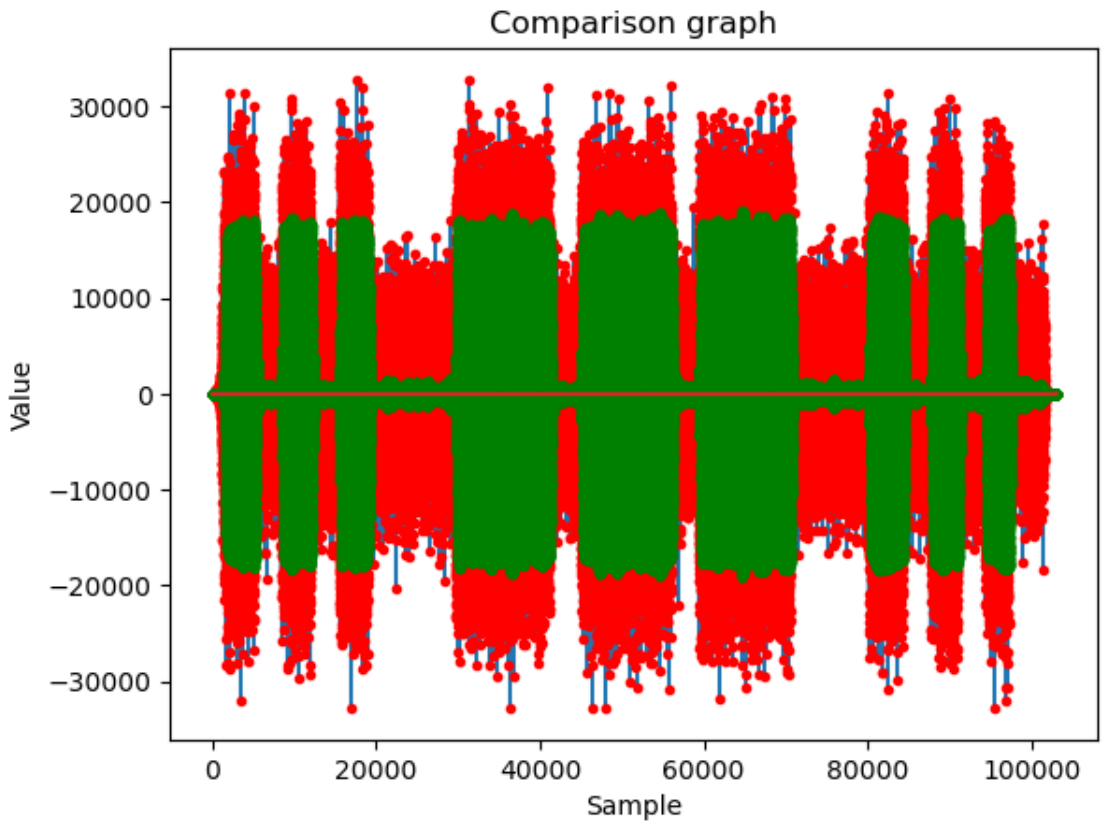


[ ]: <Figure size 640x480 with 0 Axes>

<Figure size 640x480 with 0 Axes>

Q 5.3

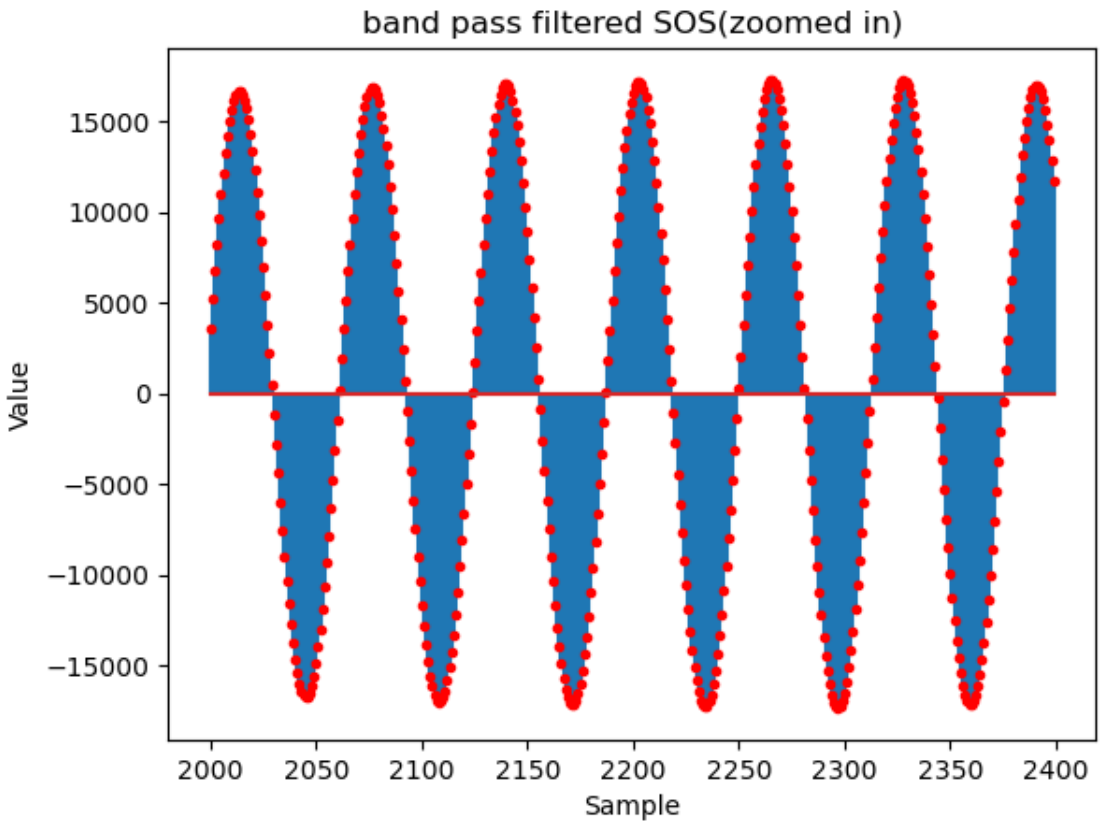
[63]:
```
sos_filtered_5 = signal.convolve(sos_noisy,band_pass)
plt.figure()
plt.stem(sos_noisy,markerfmt='r.')
plt.stem(sos_filtered_5,markerfmt='g.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Comparison graph')
plt.show()
```

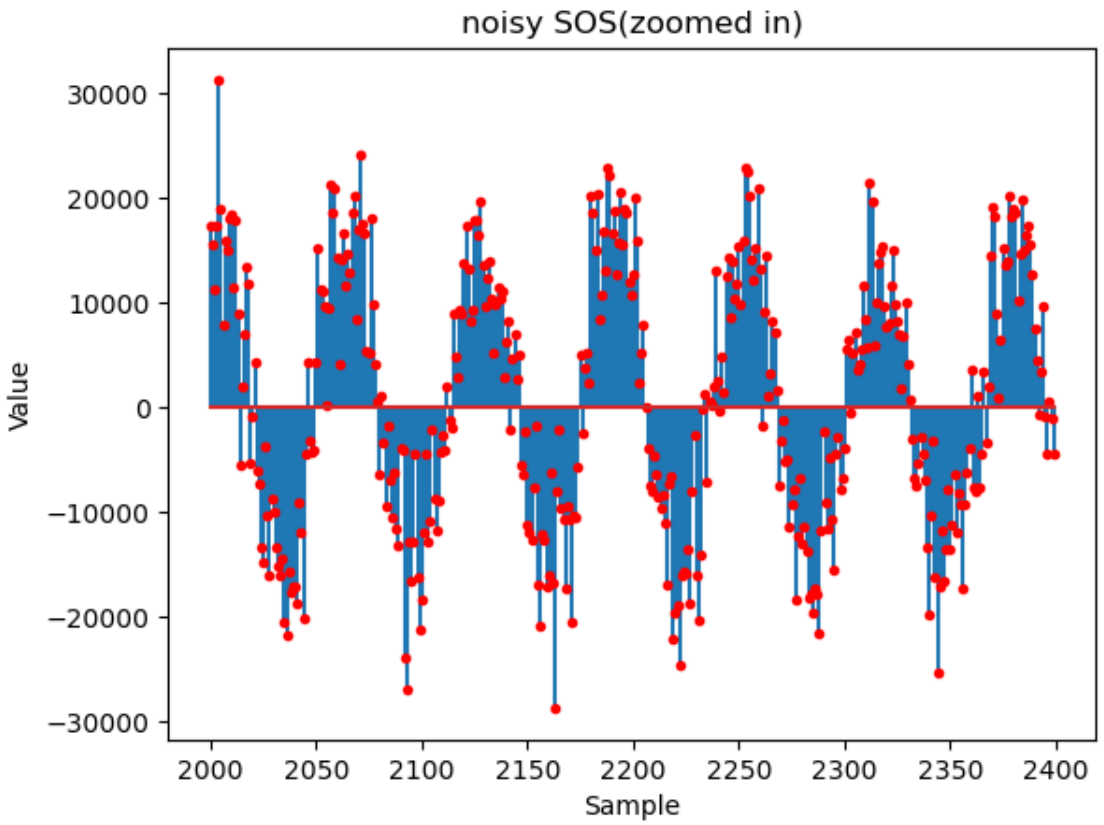


When plotting the two graphs on eachother, we can see that after applying the band pass filter, the signal has very little noise when it is supposed to be silent and is very close in shape to the clean SOS.

The amplitude of the filtered signal is also very close to the original SOS.

[65]:
```
plt.figure()
plt.stem(t,sos_filtered_5[t],markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('band pass filtered SOS(zoomed in)')
plt.show()

plt.figure()
plt.stem(t,sos_noisy[t],markerfmt='r.')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('noisy SOS(zoomed in)')
plt.show()
```


band pass filtered SOS(zoomed in)
15000
10000
5000
0
−5000
−10000
−15000
Value
2000
2050
2100
2150
2200
2250
2300
2350
2400
Sample



When comparing the zoomed in signals, it can be seen that the filtered signal almost perfectly approximates a sine wave, however it is slightly shifted to the right, making the error comparison used before unfair.

[66]:
```
ipd.display(ipd.Audio(data=sos_filtered_5, rate=fs))
ipd.display(ipd.Audio(data=sos_noisy, rate=fs))
```

```
<IPython.lib.display.Audio object>
```

```
<IPython.lib.display.Audio object>
```

When listening to the signals, one may observe that the filtered signal sounds almost exactly like the clean SOS signal, while the noisy signal is very distorted and loud.

From all of the above it can be noticed that this filter almost perfectly removes the noise from this SOS signal.