Fourier Transform and Sounds

September 29, 2020

1 Sound Test

[1]:
```python
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

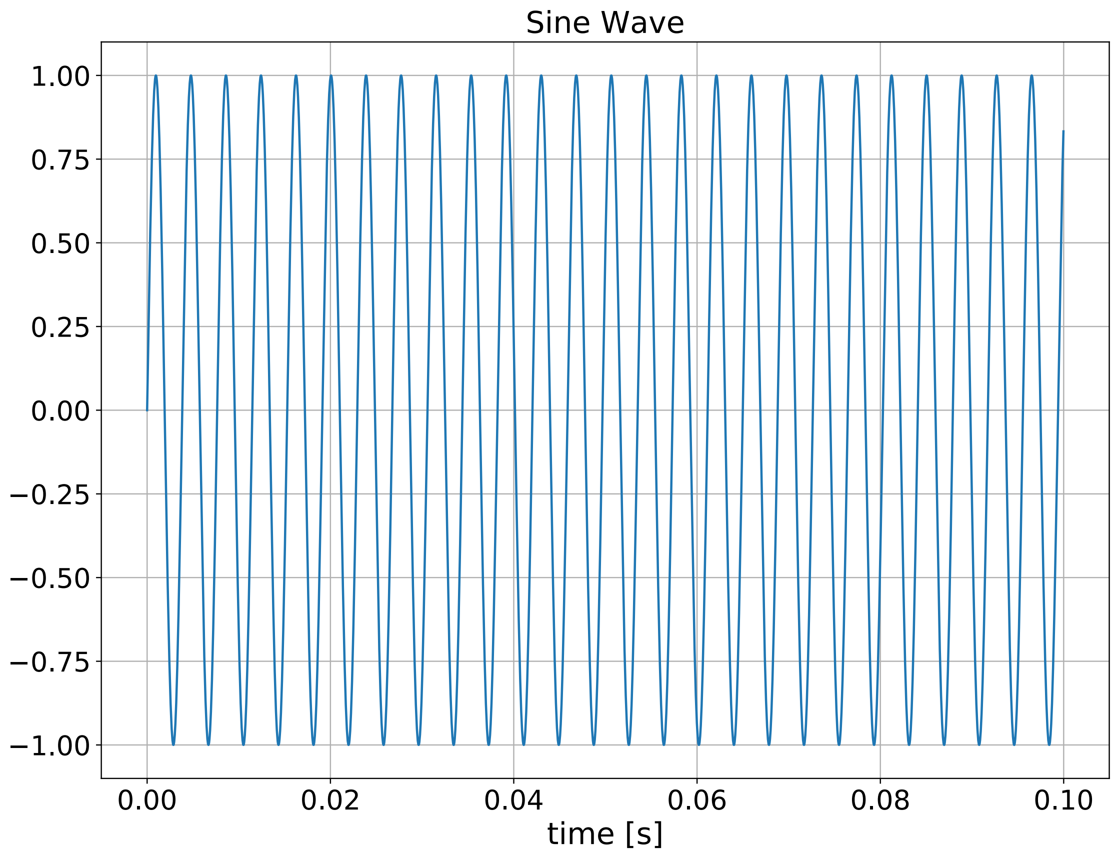
[1]:
```python
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

[2]:
```python
# play a noise sound

fs = 44100
data = 0.1*np.random.uniform(-1, 1, fs)
sd.play(data, fs)
```

[3]:
```python
# stop the sound
sd.stop()
```

2 Sine Wave: $\sin(2\pi f t), \quad f = 261.626\,$Hz (C4)

[4]:
```python
fs = 44100       # sampling rate, Hz, must be integer
duration = 2.0   # in seconds, may be float
f = 261.626       # frequency, Hz,  C4:261.626,   A4:440.0
t = np.arange(fs*duration) / fs

# generate samples, note conversion to float32 array
fct = np.sin(2 * np.pi * f * t)
samples = fct.astype(np.float32)

# samples for 0.2 seconds
samples2 = samples[:4410]

plt.figure(figsize=(12,9))
plt.title('Sine Wave', fontsize=20)
plt.xlabel('time [s]',fontsize=20)
# plt.ylabel('y', fontsize=20)
plt.tick_params(labelsize=18)
plt.plot(t[:len(samples2)], samples2)
plt.grid(True)
plt.show()
```



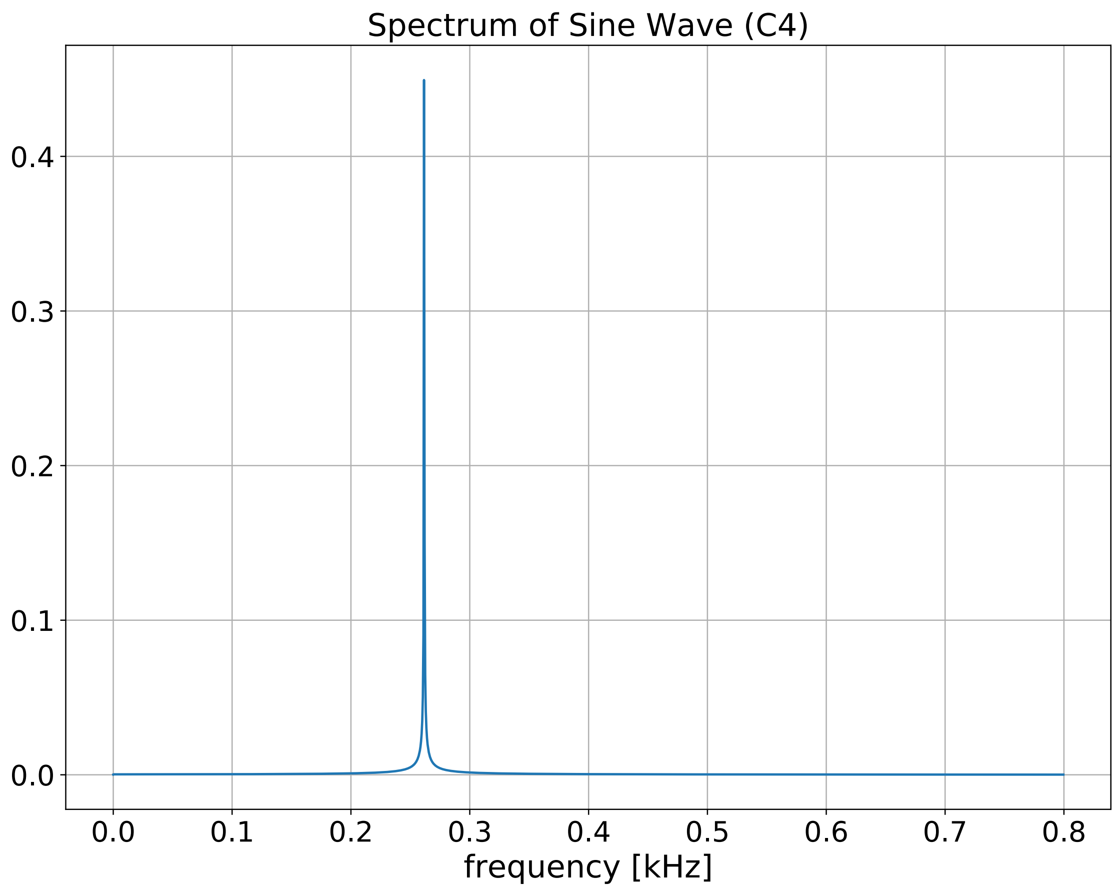
[5]:
```
# play the sound of sine wave
sd.play(0.1*samples, fs)
```

[6]:
```
sd.stop()
```

2.1 Spectrum Obtained by FFT

[7]:
```
nsamples = np.int(fs * duration)

freq = np.linspace(0, 22.05, nsamples//2)
F = np.fft.fft(samples)
F = F / nsamples
```

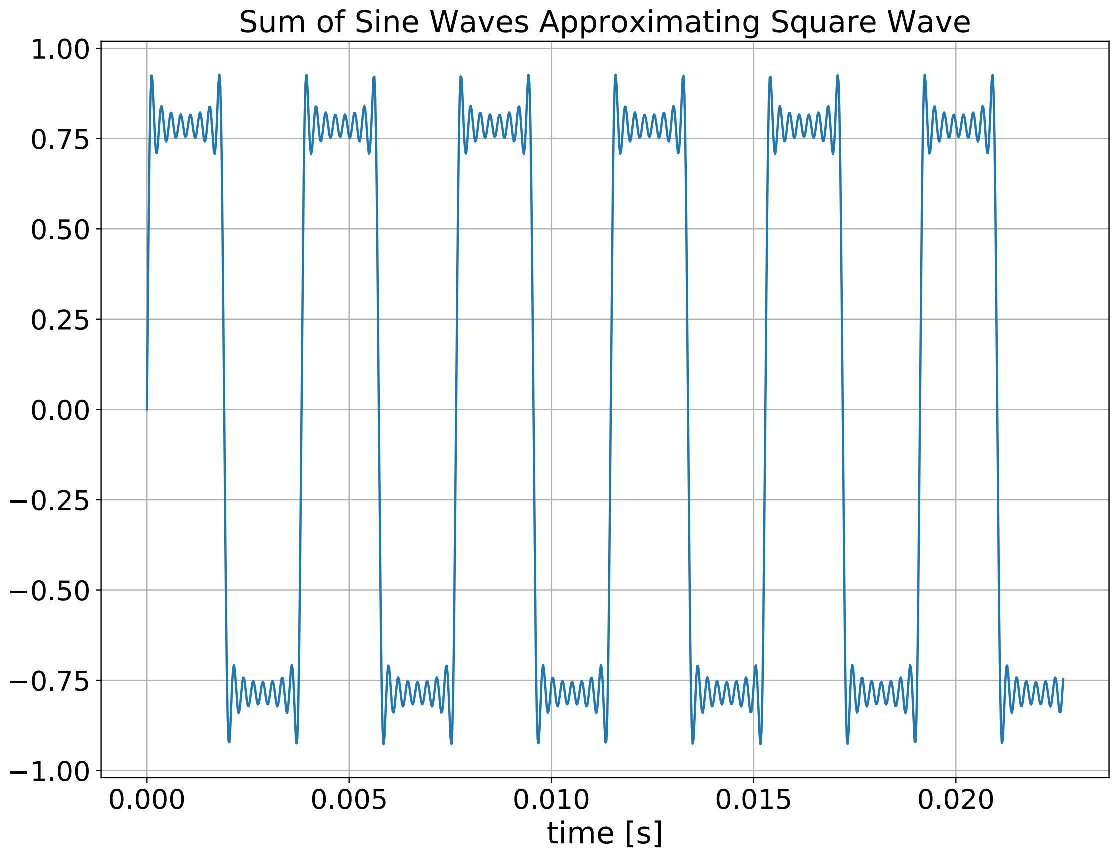
```
F[0] = F[0] / 2

samples2 = abs(F)[:1600]

plt.figure(figsize=(12, 9))
plt.title('Spectrum of Sine Wave (C4)', fontsize=20)
plt.xlabel('frequency [kHz]', fontsize=20)
# plt.ylabel('y', fontsize=20)
plt.tick_params(labelsize=18)
plt.plot(freq[:len(samples2)], samples2)
plt.grid(True)
plt.show()
```



3 Sums of Sine Waves

3.1 $\sum_{n=0}^{7} \frac{1}{2n+1} \sin\Big((2n+1) \cdot 2\pi f t\Big)$

[8]:
```
# sum of sine waves
t = np.arange(fs*duration) / fs
fct = sum(1/(2*n+1)*np.sin((2*n+1)*2*np.pi*f*t) for n in range(8))
samples = fct.astype(np.float32)
samples2 = samples[:1000] # 0.0226 s

plt.figure(figsize=(12, 9))
plt.title('Sum of Sine Waves Approximating Square Wave', fontsize=20)
plt.xlabel('time [s]', fontsize=20)
# plt.ylabel('y', fontsize=20)
plt.tick_params(labelsize=18)
plt.plot(t[:len(samples2)], samples2)
plt.grid(True)
plt.show()
```

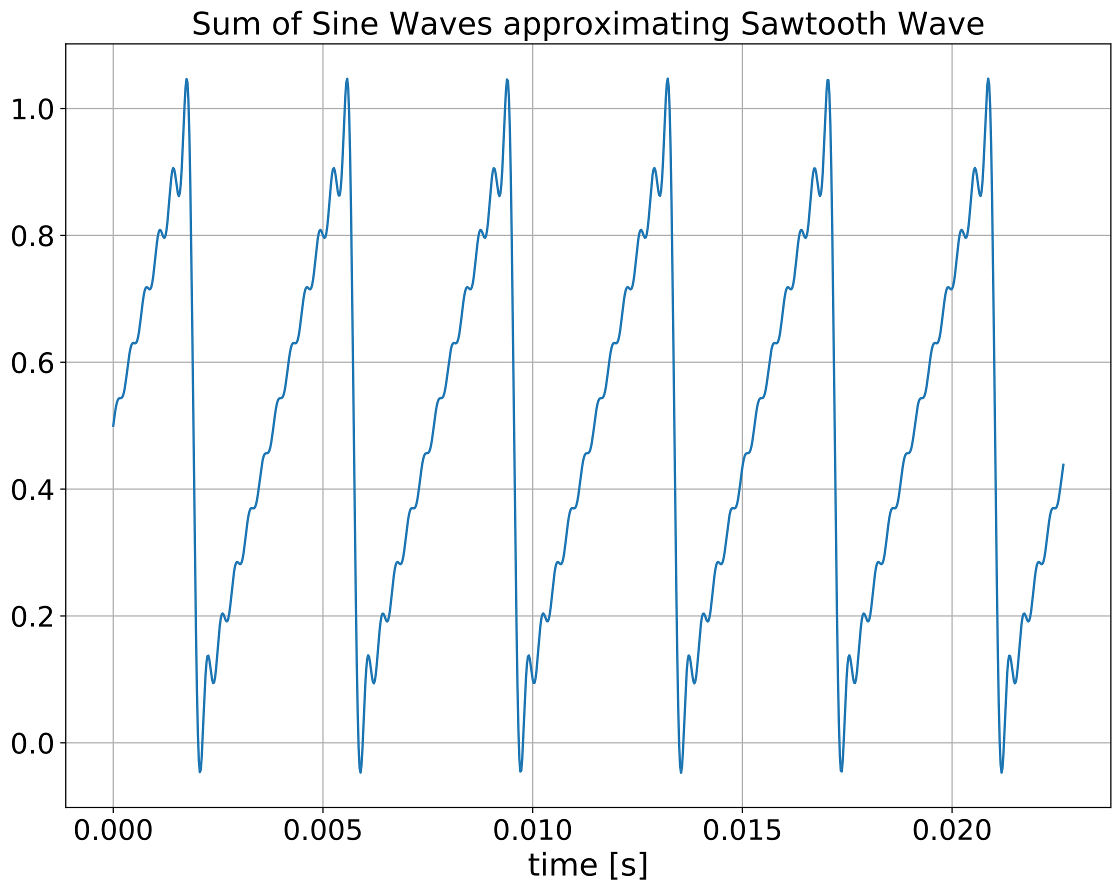


[9]:
```
# play the sound
sd.play(0.2*samples, fs)
```

[10]:
```
sd.stop()
```

3.2 $\frac{1}{2} - \frac{1}{\pi}\sum_{n=1}^{11}\frac{(-1)^n}{n}\sin\left(n\cdot 2\pi ft\right)$

```
[11]: # sum of sine wave approximating sawtooth wave
      t = np.arange(fs*duration) / fs
      fct = 0.5 - 1/np.pi*sum((-1)**n/n*np.sin(n*2*np.pi*f*t) for n in range(1, 12))
      samples = fct.astype(np.float32)
      samples2 = samples[:1000]

      plt.figure(figsize=(12, 9))
      plt.title('Sum of Sine Waves approximating Sawtooth Wave', fontsize=20)
      plt.xlabel('time [s]', fontsize=20)
      # plt.ylabel('y', fontsize=20)
      plt.tick_params(labelsize=18)
      plt.plot(t[:len(samples2)], samples2)
      plt.grid(True)
      plt.show()
```



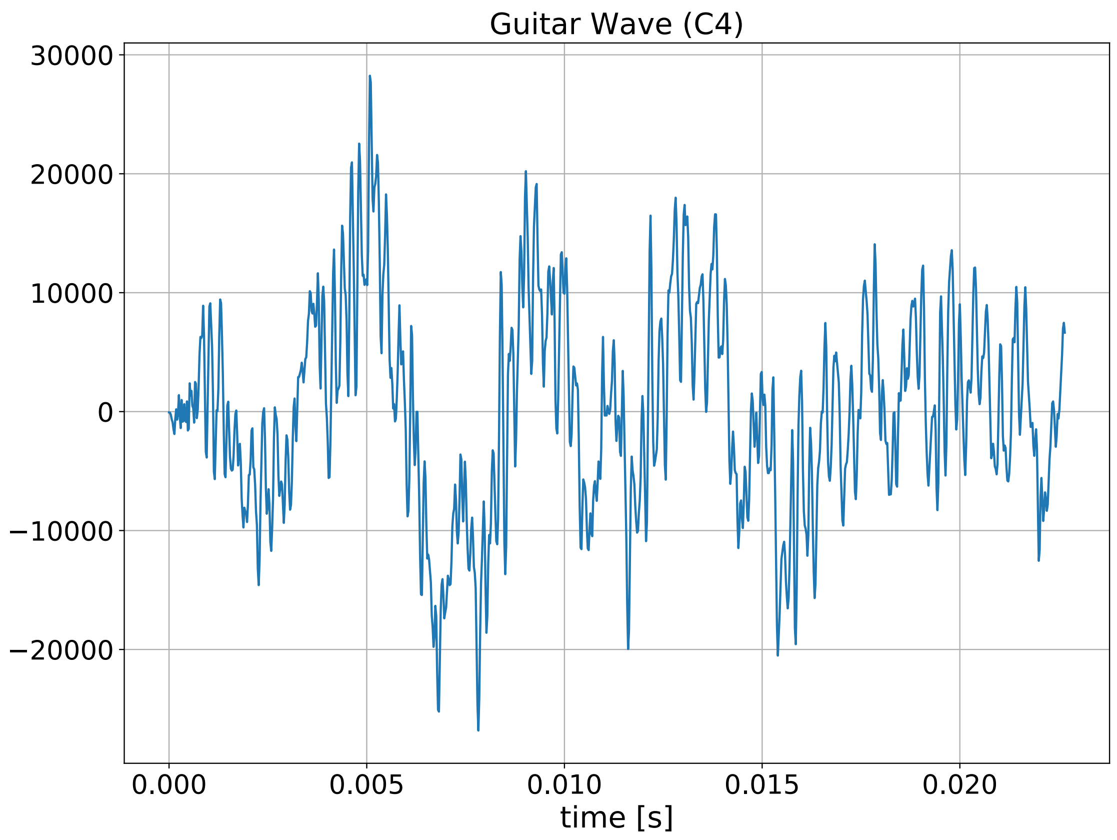
```
[12]: sd.play(0.2*samples, fs)
```

[13]:
```
sd.stop()
```

4 Wave Form of Guitar Sound

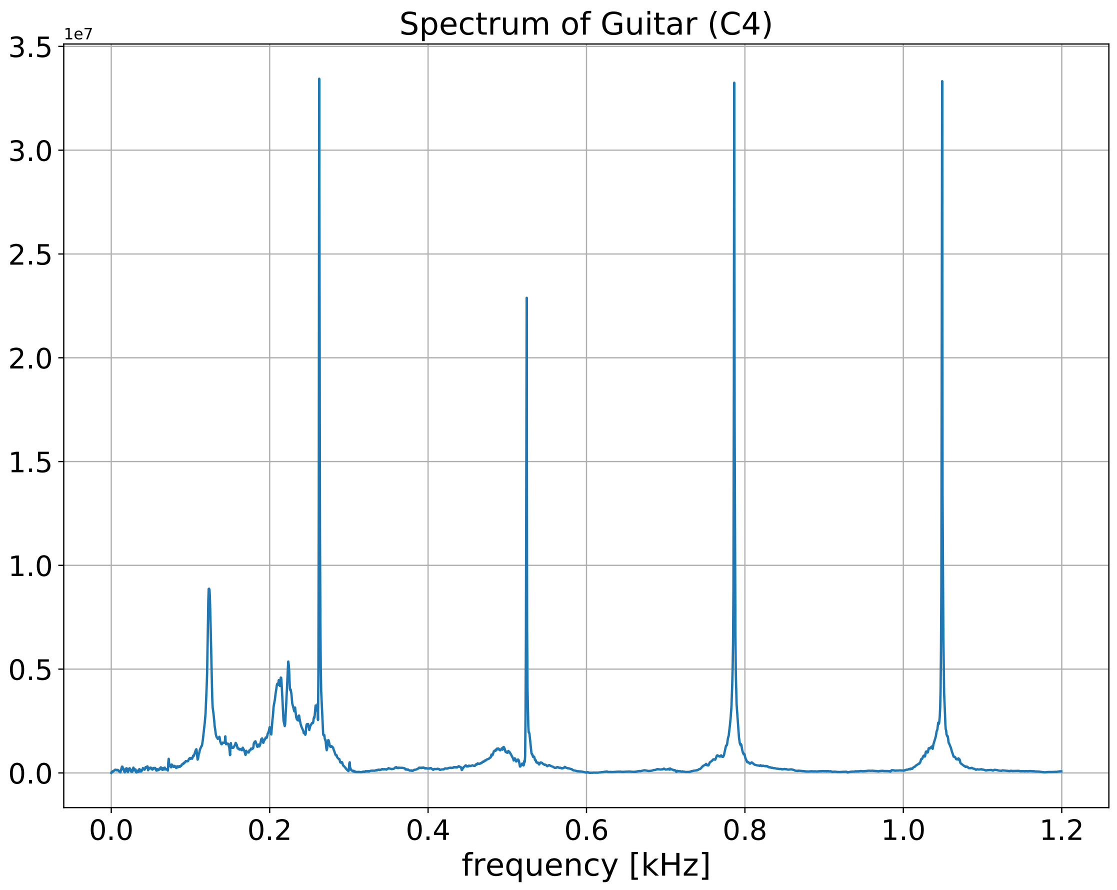
[14]:
```
filename = "Guitar-C4.wav" # sampling rate: 44.1 kHz, 2ch
nsamples = np.int(88200) # 2 sec.
rate, data = wavfile.read(filename)
data0 = data[:nsamples,0] # ch. 0
```

[15]:
```
data0_ = data0[:1000]

plt.figure(figsize=(12, 9))
plt.title("Guitar Wave (C4)", fontsize=20)
plt.xlabel("time [s]", fontsize=20)
# plt.ylabel('y', fontsize=20)
plt.tick_params(labelsize=18)
plt.plot(t[:len(data0_)], data0_)
plt.grid(True)
plt.show()
```



5 Spectrum of Guitar Sound

[15]:
```
freq = np.linspace(0, 22.05, nsamples//2)
F = np.fft.fft(data0)
# F = F / (nsamples/2)
F[0] = F[0] / 2
F_half = F[:len(F)//2]

F_ = abs(F_half)[:2400]

plt.figure(figsize=(12, 9))
plt.title('Spectrum of Guitar (C4)', fontsize=20)
plt.xlabel('frequency [kHz]', fontsize=20)
# plt.ylabel('y', fontsize=20)
plt.tick_params(labelsize=18)
plt.plot(freq[:len(F_)], F_)
plt.grid(True)
plt.show()
```



6 Easy High-pass-filter by FFT

[16]:
```
t = np.linspace(0, nsamples/44100, nsamples)
freq = np.linspace(0, 22.05, nsamples//2)

F = np.fft.fft(data0)
F[0] = F[0] / 2
F_half = F[:len(F)//2]
```

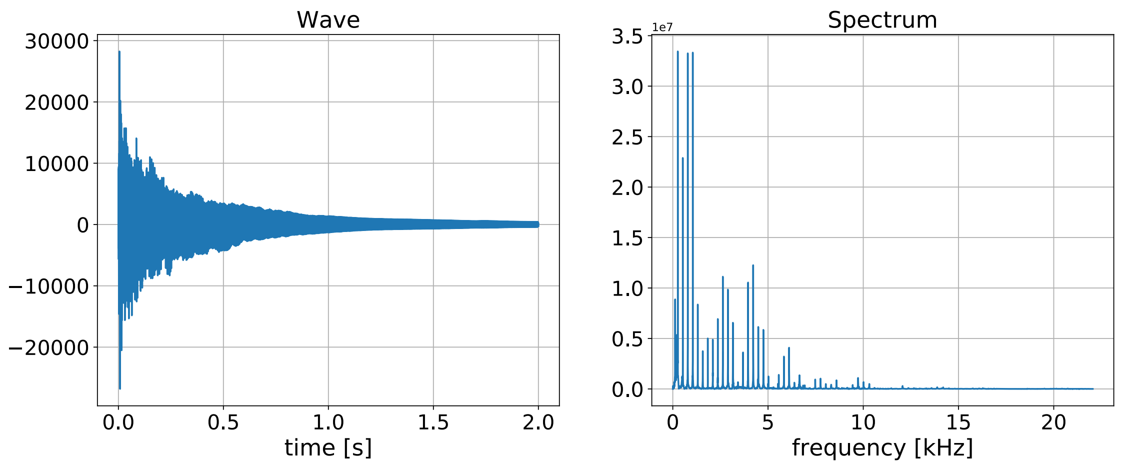
Set the cutoff frequency 0.6 kHz

[17]:
```
F2 = F.copy()
F2_half_0, F2_half_1 = F2[:nsamples//2], F2[nsamples//2:]
# F2_half_0.shape, F2_half_1.shape
cutoff = 0.6
F2_half_0[(freq < cutoff)] = 0
F2_half_1[(freq > 22.05-cutoff)] = 0
F2 = np.concatenate([F2_half_0, F2_half_1])
```

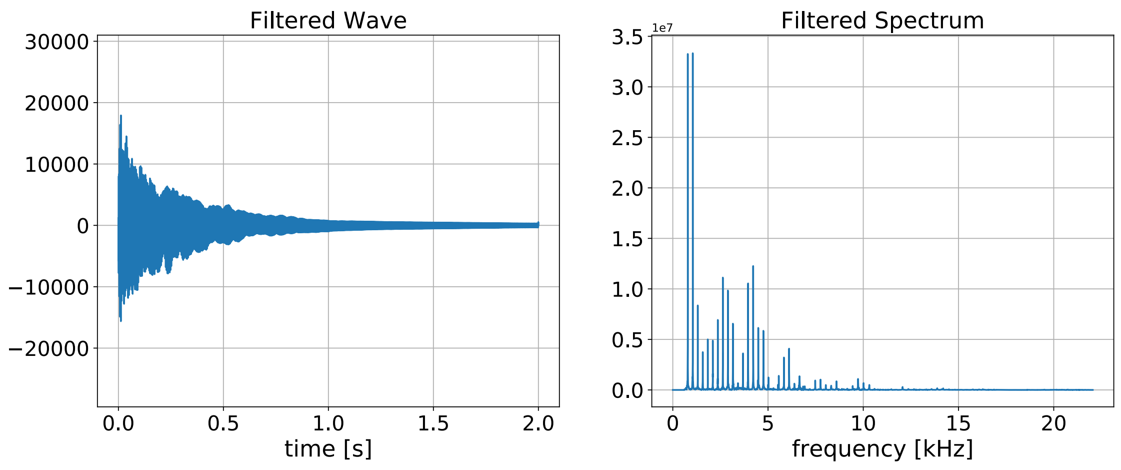
[18]:
```
data0_fil = np.fft.ifft(F2)
data0_fil = np.real(data0_fil) # * nsamples) # to original scaling

# plt.rcParams['font.family'] = 'Dejavu Sans'

plt.figure()

fig, (L1, R1) = plt.subplots(ncols=2, figsize=(16,6), sharex=False)

L1.set_title('Wave', fontsize=20)
L1.set_xlabel('time [s]', fontsize=20)
#L1.set_ylabel('amp', fontsize=20)
L1.tick_params(labelsize=18)
L1.plot(t, data0)
L1.grid(True)
R1.set_title('Spectrum', fontsize=20)
R1.set_xlabel('frequency [kHz]', fontsize=20)
R1.tick_params(labelsize=18)
R1.plot(freq, abs(F_half))
R1.grid(True)
ylim_l = L1.get_ylim()
ylim_r = R1.get_ylim()

plt.figure();
fig2, (L2, R2) = plt.subplots(ncols=2, figsize=(16,6), sharex=False)
L2.set_title('Filtered Wave', fontsize=20)
L2.set_xlabel('time [s]', fontsize=20)
L2.tick_params(labelsize=18)
```

```
L2.set_ylim(ylim_l)
L2.plot(t, data0_fil)
L2.grid(True)
R2.set_title('Filtered Spectrum', fontsize=20)
R2.set_xlabel('frequency [kHz]', fontsize=20)
R2.tick_params(labelsize=18)
R2.set_ylim(ylim_r)
R2.plot(freq, abs(F2_half_0))
R2.grid(True)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



```
<Figure size 432x288 with 0 Axes>
```



[ ]:

```

```

[19]:

```
# play the original sound
sd.play(data0, 44100)
```

[20]:

```
# play the filterd sound
sd.play(np.int16(np.round(data0_fil)), 44100)
```

[21]:

```
sd.stop()
```