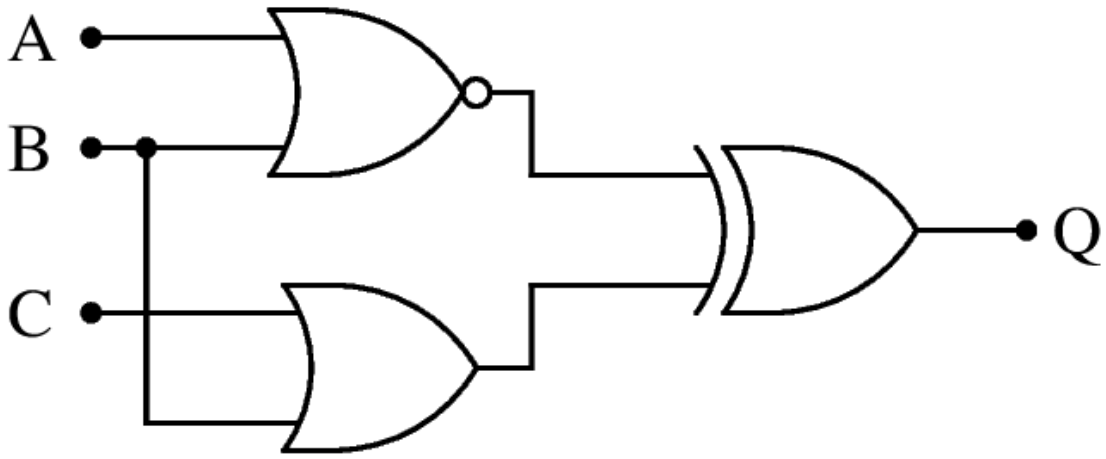
Digital-Data-Logic-Lab-Solution

March 4, 2023

1 Lab 04: Digital Data & Logic Solutions

1.1 Logic



A	B	C	Q
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

1.2 Binary Search Algorithms / Successive Approximation ADC

[72]:
```
# number of bits
max_n = 10000000
min_n = 10000
precision = 1000
discrete_values = (max_n - min_n) / precision
print(f'Discrete Values: {discrete_values}')
```

```
num_bits = int(np.ceil(np.log2(discrete_values)))
print(f"Min Num Bits: {num_bits}")
```

```
Discrete Values: 9990.0
Min Num Bits: 14
```

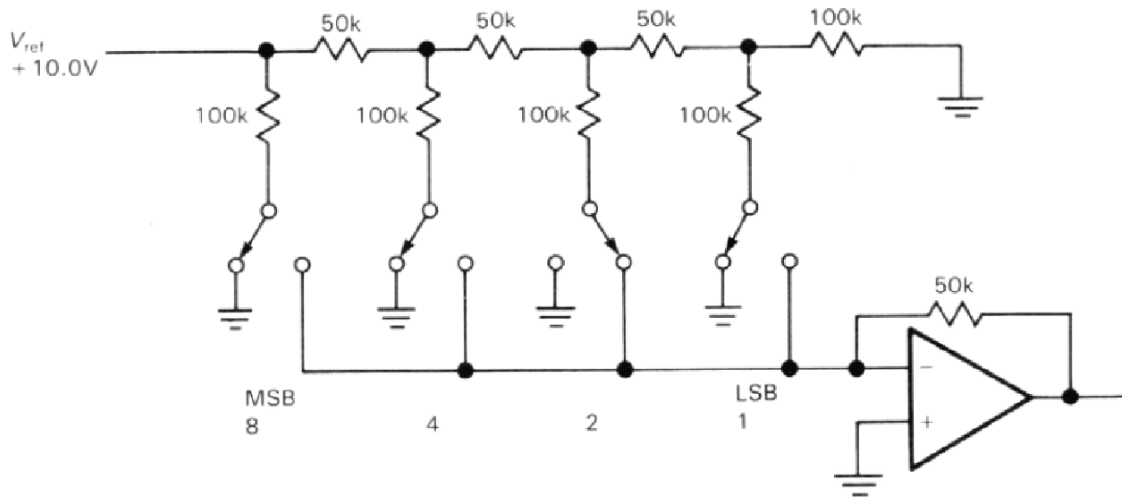
[73]:
```
import numpy as np

def saadc(base10_in, n_bits, min_10, max_10):
    """Successive approximation ADC algorithm

    Inputs:
        base10_in: base10 number
        n_bit: number of bits
        min_10: minimum base10 number to approximate
        max_10: maximum base10 number to approximate

    Returns:
        base2n: base2 approximations
        base10num: base10 value of base2 approximation
        pct_err: percent error (base10_approx:num_10)
    """
    if base10_in < min_10 or base10_in > max_10:
        raise ValueError(f'Input base10 number out of approximator range')

    print(f"Input Base10 Number: {base10_in}")
    print(f"Bit Depth: {n_bits}")

    LSB = (max_10 - min_10)/(2**n_bits - 1)
    print(f"LSB: {LSB}")

    bits = np.zeros(n_bits)

    base10approx = min_10
    for n, bit in enumerate(range(n_bits-1, -1, -1)):
        print(f"Approximating bit {bit}...")

        bits[n] = 1
        print(f'\tBase2 Approximation: {bits}')

        approx_n = base10approx + LSB*(2**bit)
        if approx_n < base10_in:
            base10approx = approx_n
        else:
            base10approx = approx_n - LSB*(2**bit)
            bits[n] = 0
            print(f'{approx_n} > {base10_in}')
```

```
        print(f'\tBase2 Approximation: {bits}')
        print(f'\tBase10 Approximation: {base10approx}')


    pct_err = 100*((base10approx - base10_in)/base10_in)
    print(f"Percent Error: {pct_err:.2f}")



saadc(3500000, 14, 10000, 10000000)
```

```
Input Base10 Number: 3500000
Bit Depth: 14
LSB: 609.7784288591834
Approximating bit 13…
        Base2 Approximation: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
5005304.88921443 > 3500000
        Base2 Approximation: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 10000.0
Approximating bit 12…
        Base2 Approximation: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base2 Approximation: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 2507652.444607215
Approximating bit 11…
        Base2 Approximation: [0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
3756478.6669108225 > 3500000
        Base2 Approximation: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 2507652.444607215
Approximating bit 10…
        Base2 Approximation: [0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base2 Approximation: [0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 3132065.5557590188
Approximating bit 9…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 3444272.111334921
Approximating bit 8…
        Base2 Approximation: [0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
3600375.3891228717 > 3500000
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 3444272.111334921
Approximating bit 7…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
3522323.7502288963 > 3500000
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 3444272.111334921
Approximating bit 6…
```

```
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 3483297.930781909
Approximating bit 5…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0.]
3502810.8405054025 > 3500000
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
        Base10 Approximation: 3483297.930781909
Approximating bit 4…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0.]
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0.]
        Base10 Approximation: 3493054.385643656
Approximating bit 3…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0.]
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0.]
        Base10 Approximation: 3497932.6130745295
Approximating bit 2…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0.]
3500371.726789966 > 3500000
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0.]
        Base10 Approximation: 3497932.6130745295
Approximating bit 1…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0.]
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0.]
        Base10 Approximation: 3499152.1699322476
Approximating bit 0…
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 1.]
        Base2 Approximation: [0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 1.]
        Base10 Approximation: 3499761.9483611067
Percent Error: -0.01
```

1.3 DAC: R-2R Ladder



1. Solve for the currents through the 2R (100 kΩ) resistors associated with each bit branch for the binary numbers `1100` and `0011`. A switch will be connected to GND when it represents a `0`. Remember that because the op amp is configured with negative feedback, `V-` = `V+` = `GND`.

2. How do the bit branch currents (1) compare to one another, and (2) change as a function of the binary digit they represent (0 or 1)? Does this make sense?

The current through each 100 $k\Omega$ resistor drops by a factor of 2 from MSB → LSB: 0.1 mA, 0.05 mA, 0.025 mA, 0.0125 mA. This makes sense since the output voltage is a function of the sum of the current that foes through the op amp feedback resistor, and each successive lesser significant bit contributes a factor of 2 less than the previous more significant bit.

3. What is Vo, the output voltage from the op amp, for the two binary numbers?

- $ 1100_2 →V_o = 7.5 V $
- $ 0011_2 →V_o = 1.875 V $

1.4 Canine ECG Data

1.4.1 HDF5 Format Data

The HDF5 data format allows for metadata to be stored along with the actual data, making data extraction and manipulation much easier. Most newer Matlab files (>= v7.3) are actually HDF5 files.

We will work with some ECG data acquired on one of my dogs. In the HDF5 file, the ECG data (`ecg`) are in millivolts and the time data (`t`) are in seconds.

[74]:
```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()

f1 = h5py.File('canine_ecg.h5','r')
```

[75]:
```
# Lets see what variables ("keys") exist in the file...
list(f1.keys())
```

[75]:
```
['ecg', 't']
```

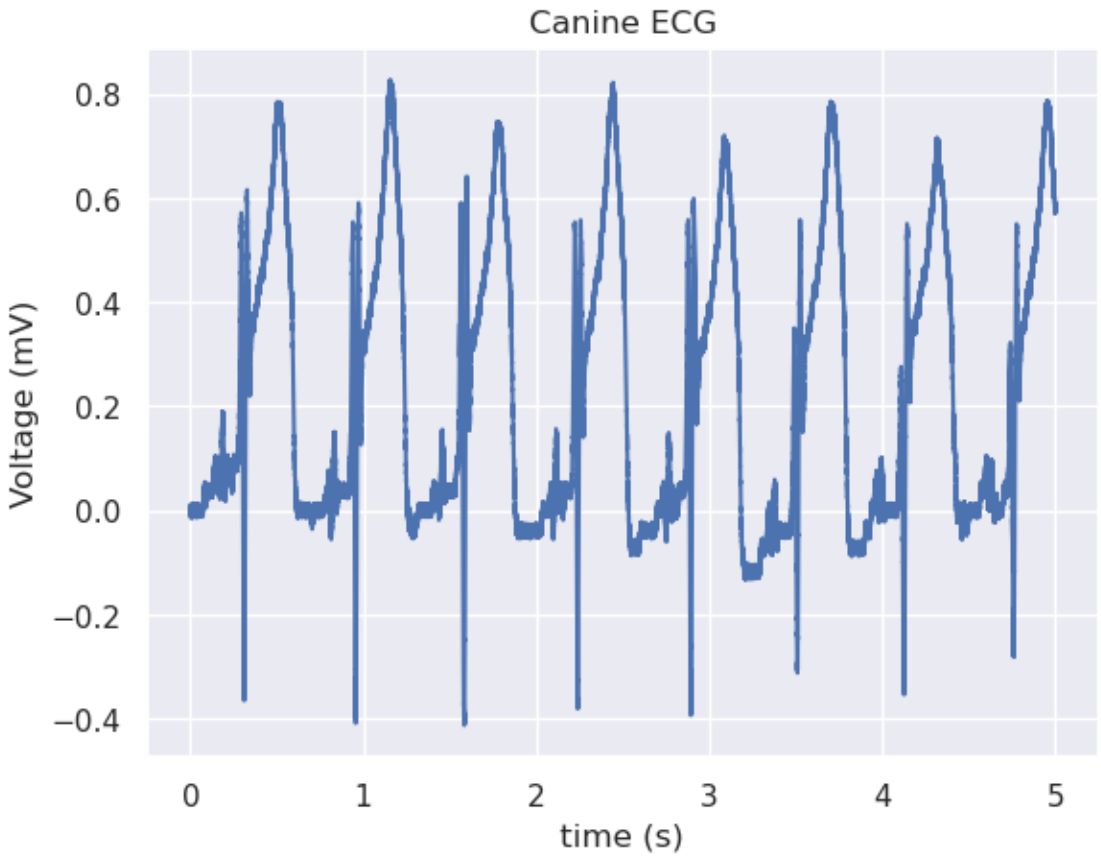
[76]:
```
# What are the dimensions of these lists...
print(f1['t'].shape)
print(f1['ecg'].shape)
```

```
(1, 100000)
(1, 100000)
```

1. Plot the ECG signal as a function of time, with the axes properly labeled with units.

[77]:
```
# Python starts arrays with index 0 (in contrast to Matlab/Fortran starting␣
 ↪with 1).
# Our arrays have all of the data packed into the first row, which can be␣
 ↪referenced as [0].
plt.plot(f1['t'][0],f1['ecg'][0])
plt.xlabel('time (s)')
plt.ylabel('Voltage (mV)')
plt.title('Canine ECG')
```

[77]:
```
Text(0.5, 1.0, 'Canine ECG')
```

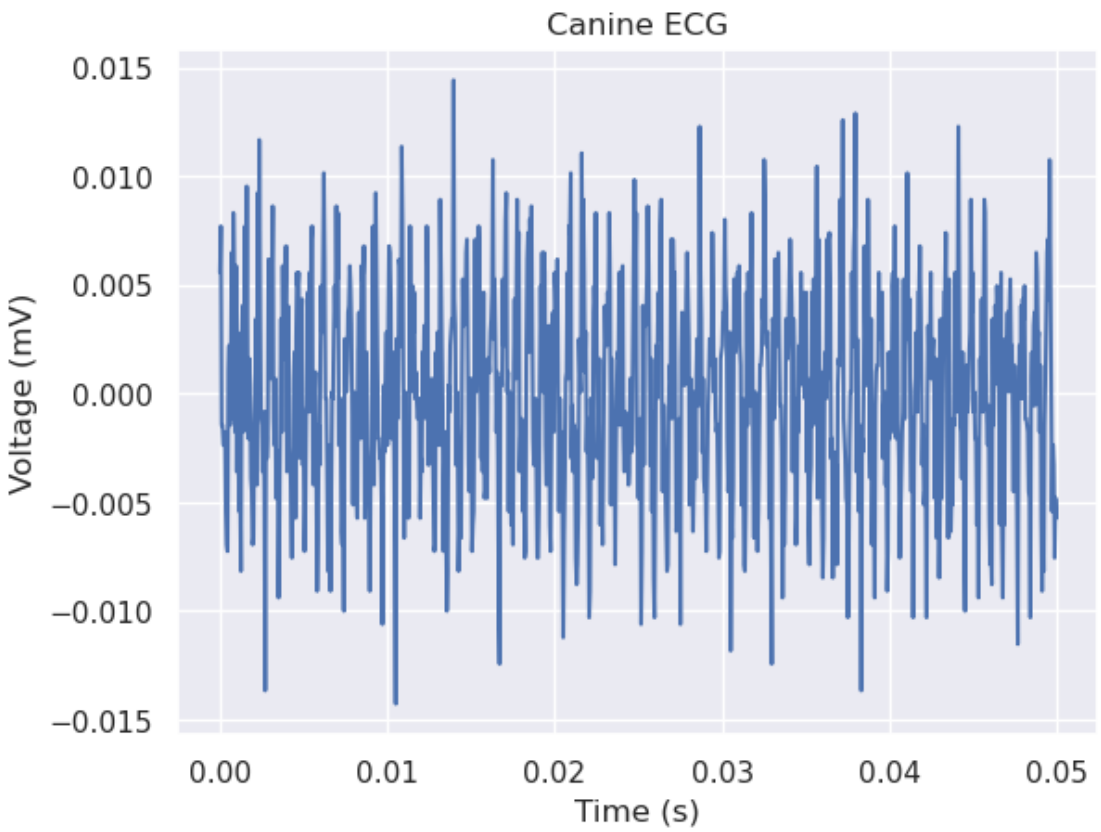


2. What is the range of voltages for the ECG signal? What is the approximate voltage range for the noise? *Note - there are several different ways that you can quantify these values that are equally valid but may provide different values. Clearly state/show the way that you estimated these values.*

[78]:
```
# inspect the noise where there should be "no" signal
plt.plot(f1['t'][0][0:1000], f1['ecg'][0][0:1000])
plt.xlabel('Time (s)')
plt.ylabel('Voltage (mV)')
```

```
plt.title('Canine ECG')
```

[78]: `Text(0.5, 1.0, 'Canine ECG')`



3. What is the minimum number of bits that we need to accurately represent the data? What was the approximate SNR for this ECG trace?

Using visual inspection of the signals: **Signal** Range (approximate): 0.8-(-0.4) = 1.2 mV

Noise Range: 0.015-(-0.015) = 0.03 mV

Minimum number of bits: 6 bits Approximate SNR: 40

$ 1.2/0.03 = 40 $ levels

$ 40 = 2^{x} $

$ x = log_{2}{40} $

$ x 5.32 $ bits

4. What should the voltage of the least-significant-bit (LSB) be set to in your ADC?

Least Significant Bit voltage: 0.01875

$\frac{1.2 \text{ mV}}{2^6} = 0.01875$ mV

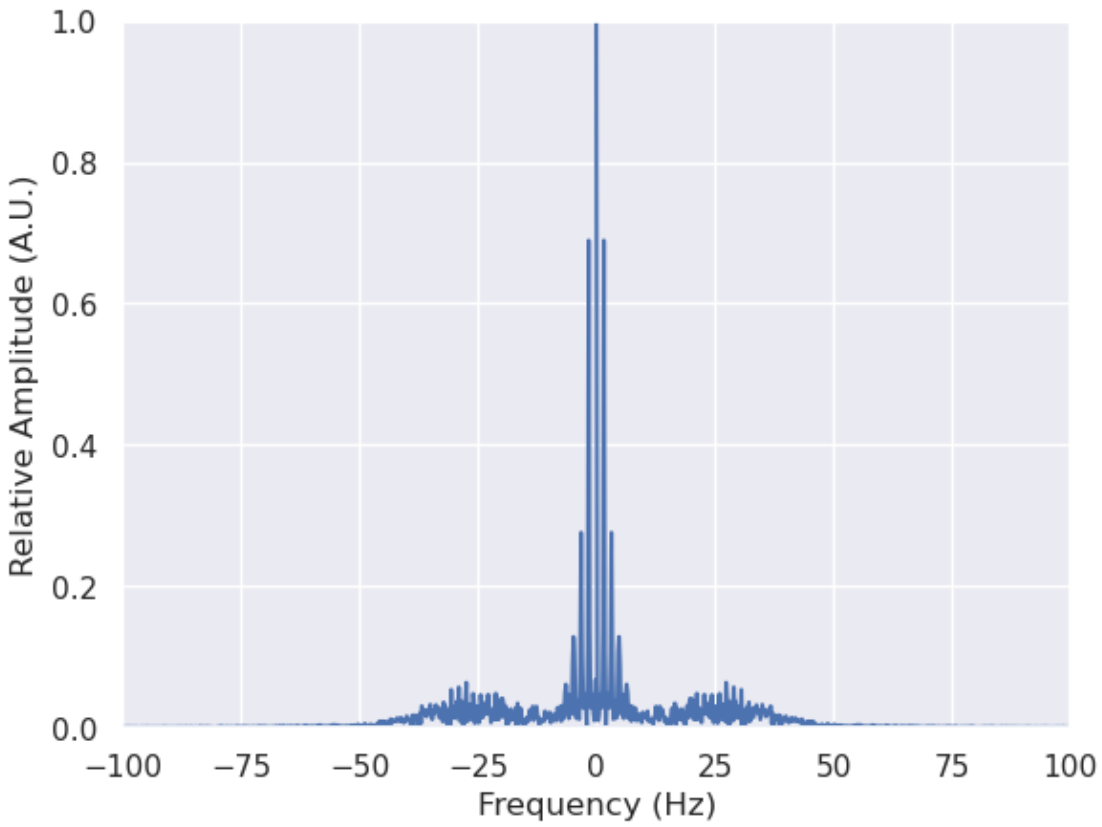
5. What is the sampling frequency (f_s) for these data?

Sample Frequency: 20000 Hz

$\frac{100000 \text{ samples}}{5 \text{ s}} = 20000$ Hz

6. Generate a plot clearly showing the frequency (spectral) content of this ECG data trace.

[79]:
```
from scipy.fft import fft, fftfreq

Fs = f1['t'].shape[1]/f1['t'][0][-1] # sample rate (samples/s)

X = fft(f1['ecg'][0])
N = len(X)

# Need this next step to properly scale and align the frequency axis
freq = fftfreq(N,1/Fs)

# Normalize the plot and zoom in on relevant content...
plt.plot(freq,np.abs(X)/np.abs(X).max())
plt.xlabel('Frequency (Hz)')
plt.ylabel('Relative Amplitude (A.U.)')
_ = plt.axis([-100, 100, 0, 1])
```



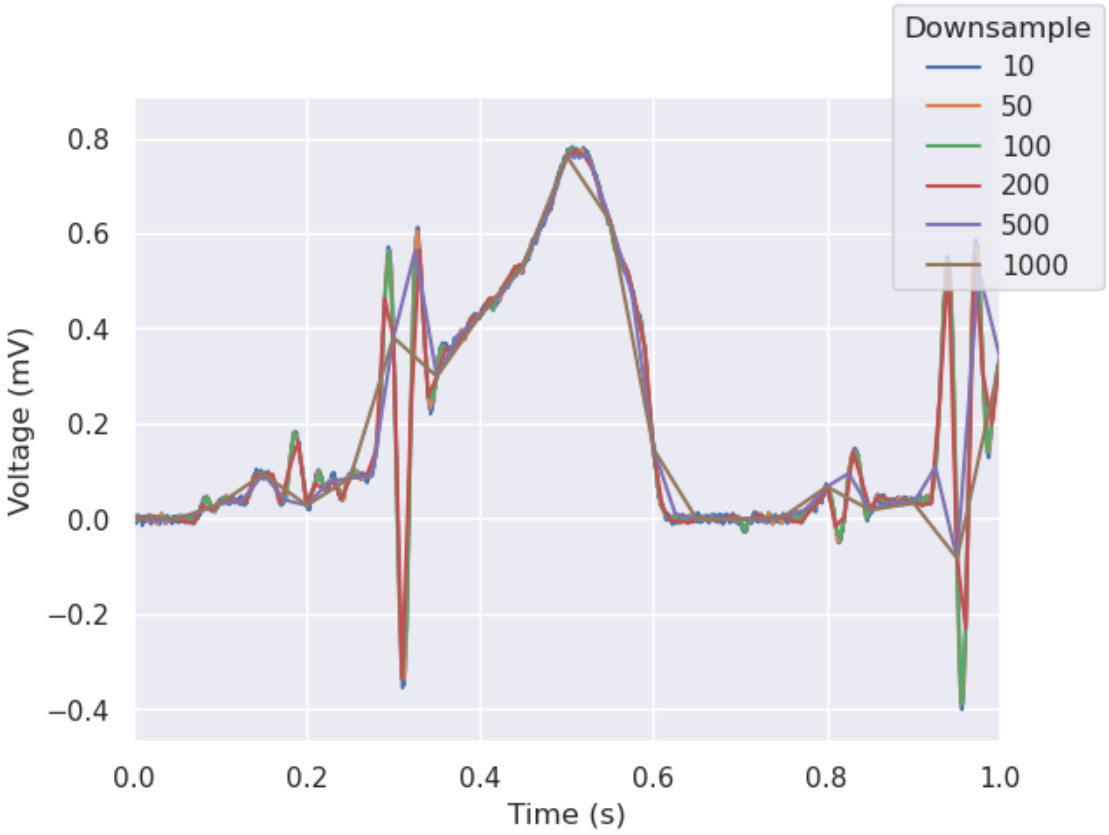
Next, we're going to manually downsample the data to qualitatively get a feel for what minimum sampling frequency can be to still accurately represent the morphology of the ECG signal. Downsample the ECG data (and the time data) by factors of 10, 50, 100, 200, 500, and 1000. Generate plots comparing these waveforms (zoom in on just one of the wave complexes) and notice the aliasing / signal corruption that occurs when the waveform is undersampled.

1.4.2 Downsampling

7. Downsampling normally reduces the the amplitude of the frequency spectrum by a factor equal to the downsample factor, and reduces the frequency window by the downsample factor. Any signal beyond the new downsampled frequency window is alised into the new signal - distoring the data. These are seen in the downsampled frequency spectra below as expected.

[80]:
```
ax = plt.figure()
for downsample in [10, 50, 100, 200, 500, 1000]:
    plt.plot(f1['t'][0][::downsample], f1['ecg'][0][::downsample],␣
 ↪label=downsample)
ax.legend(title='Downsample')
plt.xlim([0,1])
plt.xlabel('Time (s)')
plt.ylabel('Voltage (mV)')
```

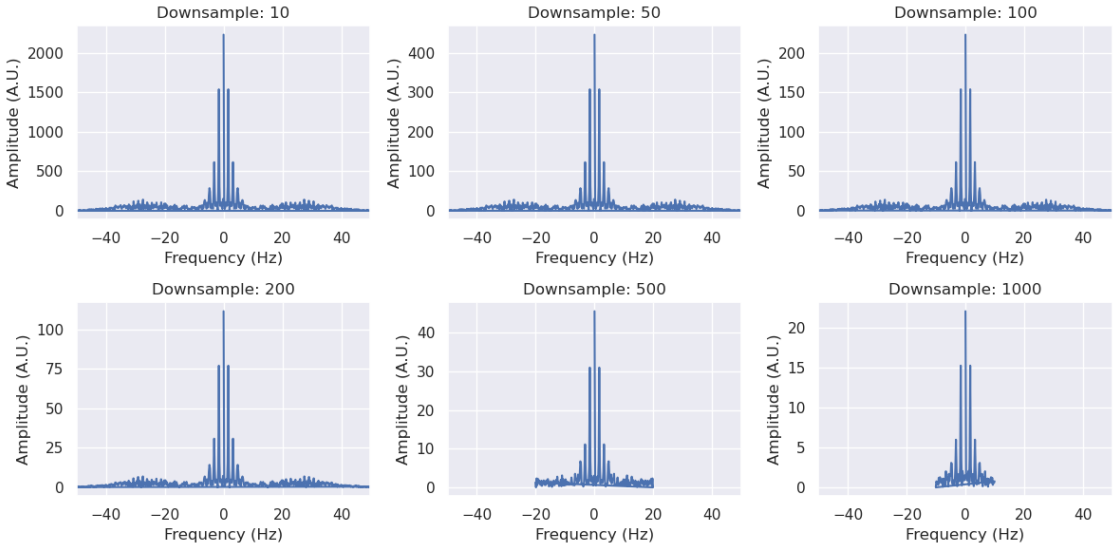
[80]:
```
Text(0, 0.5, 'Voltage (mV)')
```



8. Generate plots of the frequency content of these downsampled ECG data traces and comment on what you expected these to look like relative to the original signal and if your plots match your intuition.

[81]:
```
f = plt.figure(figsize=(12, 6))
gs = f.add_gridspec(2, 3)
for n, downsample in enumerate([10, 50, 100, 200, 500, 1000]):
    t = f1['t'][0][::downsample]
    Fs = t.shape/t[-1] # sampling rate (samples/s)

    X = fft(f1['ecg'][0][::downsample])
    N = len(X)

    freq = fftfreq(N,1/Fs)

    ax = f.add_subplot(gs[n])
    plt.plot(freq, np.abs(X), label=downsample)
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude (A.U.)')
    plt.xlim([-50, 50])
    plt.title(f'Downsample: {downsample}')
```

```
f.tight_layout()
```



9. Based on visual inspection of your plots that show the effects of aliasing, what is the minimum sampling frequency that could be used to adequately represent these data digitally? What does this imply about the maximum frequency content of the ECG signal? How does this compare with the Nyquist limit that you would have chosen based on your FFT power spectrum?

10. How much greater than this minimum sampling frequency was the data acquisition sampling frequency? Is this adequate?

Based on the plots above, downsampling by a factor of 10 does not aliase the small frequency at about 750 Hz. This small 750 Hz component suggests the maximum freuqneyc component of the ECG signal is about 750 Hz. Downsampling by 10 yields a sampling rate of about 2000 Hz which is slightly above the nyquist limit of 1500 Hz (i.e., 750 Hz x 2). This sampling frequency is 500 Hz higher than the minimum samplign frequency, which should be adequate.

1.4.3 Raw Binary Data

Many datasets in research and industry are saved as "raw'' binary files, as opposed to ASCII files (too large) or proprietary formats. These data are 32-bit floating point numbers and were saved as a serial stream of time/voltage pairs (e.g., t_1, V_1, t_2, V_2,...).

1. Plot the ECG data trace from the binary data file and compare with the HDF5 version of the file. Note any differences (if any exist).

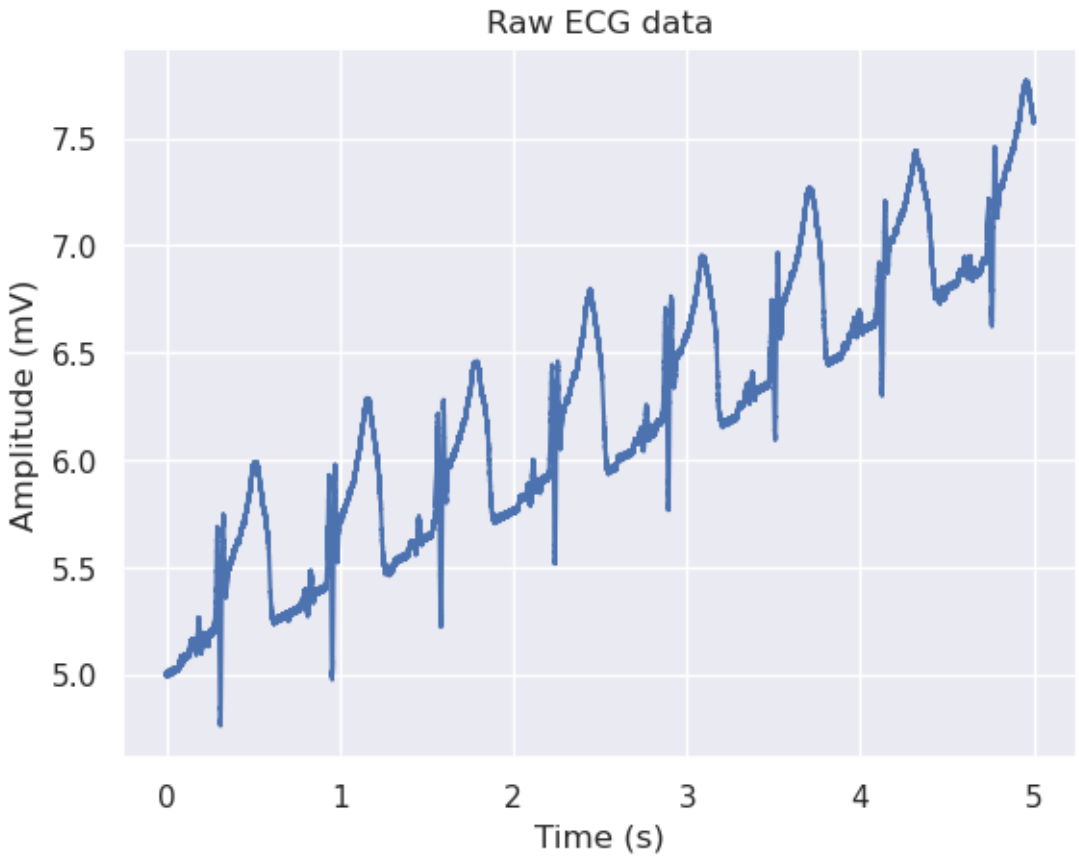
2. Comment on the pros/cons of using HDF5 vs. raw binary data for saving data.

As seen below, the data in the raw file is different from the data in the HDF5 file in a number of ways. The raw file signal drifts upwards, requires prior knowledge of the datatype (i.e. np.float32) and the organization of the data, and the zero voltage is not zero'd.

The raw file is half the size of the HDF5 file, which could attractive for larger data if storage is a concern. However, the raw file also misses vital context that would have to be provided in some other file, such as a text file. The HDF5 file can keep all relevant context into a single file.

[82]:
```
Data = np.fromfile('canine_ecg.bin',dtype=np.float32)
plt.plot(Data[::2],Data[1::2])
plt.title('Raw ECG data')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
```

[82]:
```
Text(0, 0.5, 'Amplitude (mV)')
```



Created in Deepnote