Filtering and analyzing EMG data

Noel Isaías Plascencia Díaz[1], Mitsui Myrna Salgado Saito[2], Ana Daniela del Río Pulido[3], and Erin C. McKiernan[4]

[1] Licenciatura en Física, Facultad de Ciencias, Universidad Nacional Autónoma de México
[2] Licenciatura en Ciencias de la Tierra, Facultad de Ciencias, Universidad Nacional Autónoma de México
[3] Licenciatura en Física Biomédica, Facultad de Ciencias, Universidad Nacional Autónoma de México
[4] Departamento de Física, Facultad de Ciencias, Universidad Nacional Autónoma de México

Overview

The objective of this data analysis practical is to learn how to filter EMG recordings and do some preliminary analysis. The recordings to be analyzed can be found in our GitHub repository (https://github.com/emckiernan/electrophys (https://github.com/emckiernan/electrophys)), or new recordings can be obtained by performing the 'EMG basics' laboratory practical from this series. Before carrying out this analysis practical, students should first do the 'Graphing and exploring EMG data' practical.

Setting up the notebook

We begin by setting up the Jupyter notebook and importing the Python modules for plotting figures. We include commands to view plots in the notebook, and to create figures with good resolution and large labels.

In [121]:
```
# command to view figures in Jupyter notebook
%matplotlib inline

# import plotting module
import matplotlib.pyplot as plt

# commands to create high-resolution figures with large labels
%config InlineBackend.figure_formats = {'png', 'retina'}
plt.rcParams['axes.labelsize'] = 18 # fontsize for figure labels
plt.rcParams['axes.titlesize'] = 20 # fontsize for figure titles
plt.rcParams['font.size'] = 16 # fontsize for figure numbers
plt.rcParams['lines.linewidth'] = 1.6 # line width for plotting
```

Next, we import the necesssary modules and features to perform computations, filter data, etc.

In [122]:
```
# import modules
import math
import numpy as np
import scipy as sc
from scipy import signal
from scipy.signal import butter, lfilter, filtfilt
import wave
```

Extracting and graphing the data

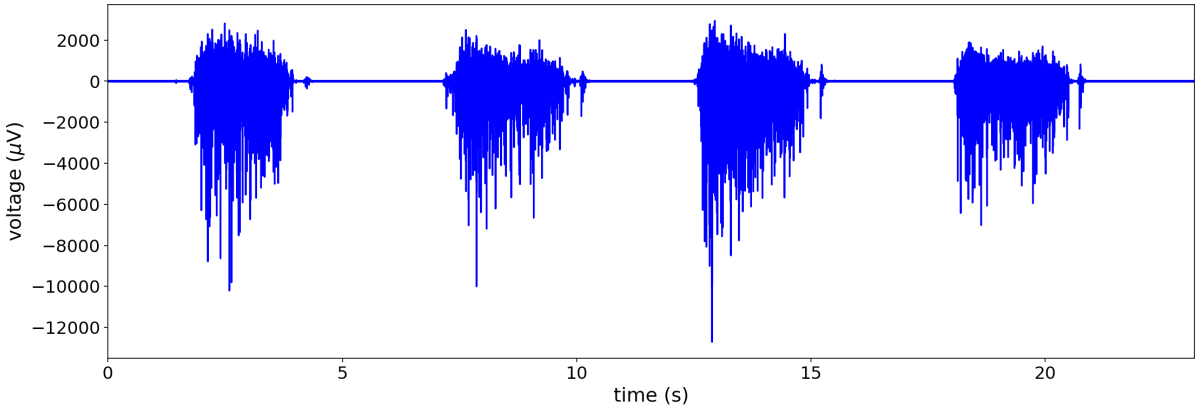
EMG recordings were obtained using the Backyard Brains EMG Spiker Box, and are saved as audio files in .wav format. We will use the same basic function as used in the 'Graphing and exploring EMG data' practical to extract and graph our data, with some extra commands to save relevant values needed for subsequent calculations.

In [123]:
```
def EMG(file):
    # open .wav file by specifying the path and filename
    record = wave.open(file)
    # extract number of channels, sample rate, data
    numChannels = record.getnchannels() # number of channels
    N = record.getnframes() # number of frames
    sampleRate = record.getframerate() # sampling rate
    # extract data from the .wav file
    dstr = record.readframes(N * numChannels)
    waveData = np.frombuffer(dstr, np.int16)
    # print the number of channels and sample rate
    print('The recording has %d channel(s).' % (numChannels))
    print('The sampling rate of the recording is %d Hz.' % (sampleRate))
    # calculate time window
    timeEMG=np.linspace(0, len(waveData)/sampleRate, num=len(waveData))
    # calculate frequency
    freq = 1/np.mean(np.diff(timeEMG))
    # save relevant data in array
    xx={'sampleRate':sampleRate,\
        'waveData':waveData,\
        'timeEMG':timeEMG,\
        'freq':freq}
    # plot EMG
    plt.figure(figsize=(18,6))
    plt.xlabel(r'time (s)')
    plt.ylabel(r'voltage ($\mu$V)')
    plt.plot(xx['timeEMG'],xx['waveData'], 'b')
    plt.xlim(0,max(xx['timeEMG']));

    return xx
```

Using our function, we can now dive into our repository of recordings. The following recording was made from the calf muscle during repeated calf raises.

In [124]:
```
xx = EMG(file='../data/S10_EMG_calf_intermittent.wav')
```

```
The recording has 1 channel(s).
The sampling rate of the recording is 44100 Hz.
```



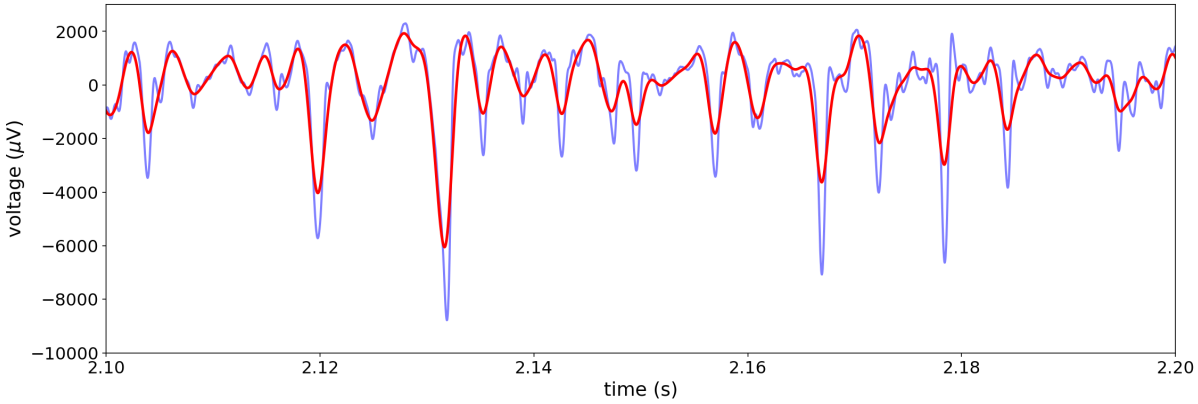
Filtering the recordings

Many EMG recordings include a certain level of noise, which can interfere with analysis. For this reason, EMG data is often filtered before proceeding. One of the most common ways to filter EMG data is by applying a band-pass filter, which preserves frequencies inside the specified range and excludes frequencies outside the range. The following functions are used to pass our raw data through a band-pass filter at 10-400 Hz.

In [125]:
```
# band-pass Butterworth filter at 10-400Hz
b, a = butter(2, ([10, 400]/(xx['freq']/2)), btype = 'bandpass')
dataf = filtfilt(b, a, xx['waveData'])
```

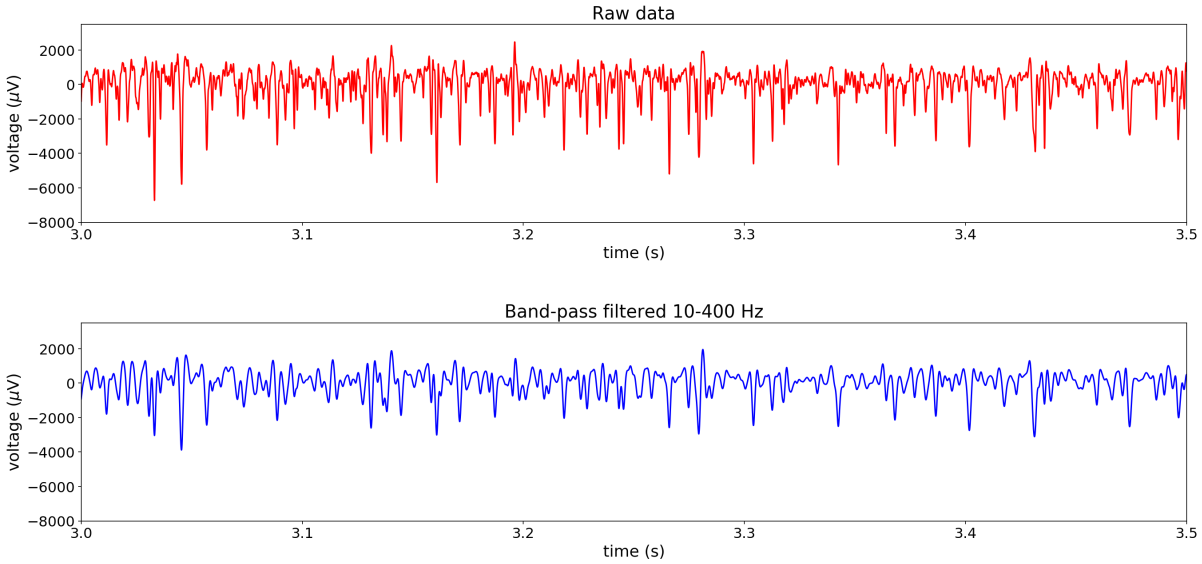
Plotting a small time window (100 ms) with the raw (blue) and filtered (red) recordings overlapping, we can see the effects of the band-pass filtering.

In [126]:
```
# plot EMG signal
plt.figure(figsize=(18,6))
plt.xlabel(r'time (s)')
plt.ylabel(r'voltage ($\mu$V)')
plt.plot(xx['timeEMG'],xx['waveData'], 'b', linewidth=2.0, alpha=0.5)
plt.plot(xx['timeEMG'],dataf, 'r', linewidth=2.5)
plt.xlim(2.1,2.2)
plt.ylim(-10000,3000);
```



Or, we can plot a larger time window (500 ms) with the recordings in separate panels to see other effects of the filtering.

In [127]:
```
fig, (ax1,ax2) = plt.subplots(2, 1, sharex= False, sharey= True, figsize
= (20,10))

# plot raw data
ax1.plot(xx['timeEMG'], xx['waveData'], 'r')
ax1.set_title('Raw data', fontsize=20)
ax1.set_xlim(3.0,3.5)
ax1.set_ylim(-8000,3500)
ax1.set_xlabel('time (s)')
ax1.set_ylabel('voltage ($\mu$V)')

# plot filtered data
ax2.plot(xx['timeEMG'], dataf, 'b')
ax2.set_title('Band-pass filtered 10-400 Hz', fontsize=20)
ax2.set_xlim(3.0,3.5)
ax2.set_ylim(-8000,3500)
ax2.set_xlabel('time (s)')
ax2.set_ylabel('voltage ($\mu$V)')

# spacing between subplots
plt.tight_layout(pad=3.0)
```



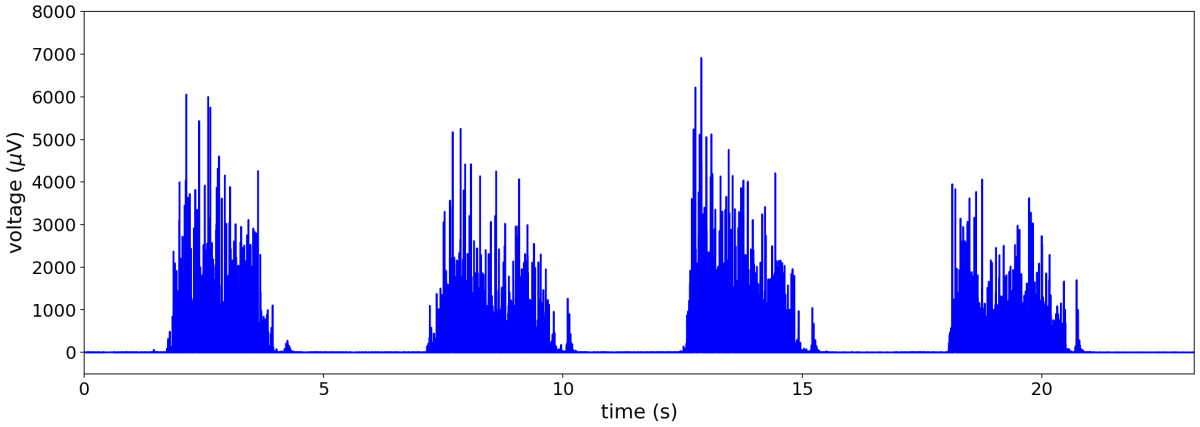
Study questions and exercises:

- Describe the effects of band-pass filtering the EMG signal.
- Why did we specify the frequency range 10-400 Hz? How is the frequency range to be used related to the physiological signal you are recording?
- What would happen if you changed the frequency band? Experiment with different filter frequency bands. Graph and compare the results.
- The recording we used as an example is clean, with very little noise. Try running the same filter on another recording from the repository which has a lower signal-to-noise ratio.
- Are there other types of filters you could use to remove noise from your recordings? Describe them and their advantages/disadvantages.

Processing EMG data

For analysis, it is sometimes easier if we rectify the EMG signal so that we end up with only positive voltage values. This will later make it possible to calculate an envelope and set a threshold for detecting contractions, among other analyses.

In [74]:
```
# rectify EMG signal
absSignal = np.absolute(dataf)
timeAbs=np.linspace(0, len(absSignal)/xx['sampleRate'], num=len(absSigna
l))

# plot rectified data
plt.figure(figsize=(18,6))
plt.xlabel(r'time (s)')
plt.ylabel(r'voltage ($\mu$V)')
plt.plot(xx['timeEMG'],absSignal,'b')
plt.xlim(0,max(xx['timeEMG']))
plt.ylim(-500,8000)
plt.show()
```

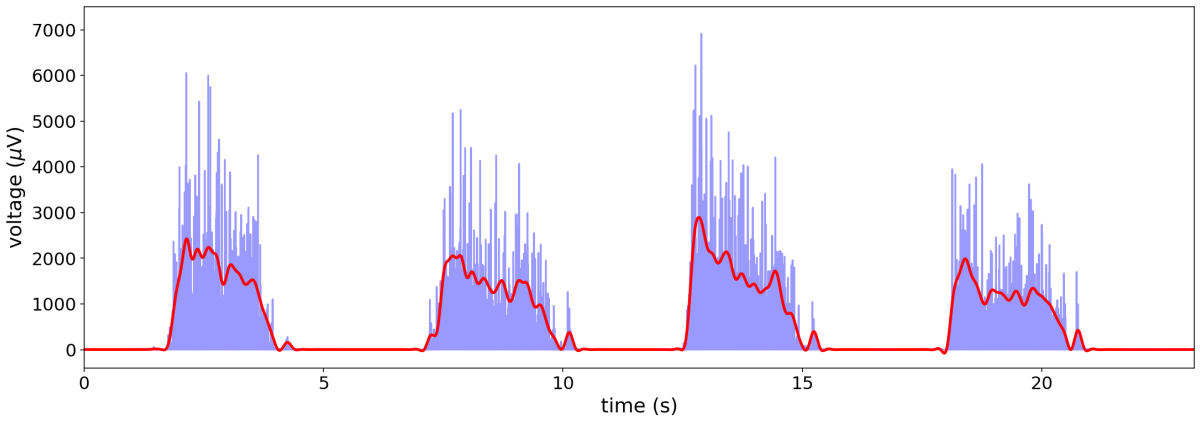


Next, we can determine the envelope of the rectified signal. One way to do this is to apply a low-pass Butterworth filter, in this case with a cutoff of 8 Hz.

In [128]:
```
# low-pass Butterworth filter for envelope detection
lowp = 8
sfreq = xx['sampleRate']
low_pass = lowp/sfreq
b, a = sc.signal.butter(4, low_pass, btype='lowpass')
datafrle = filtfilt(b, a, absSignal)
```

When we graph the data, we add a degree of transparency to the EMG signal (using the alpha parameter) and superimpose the envelope detection as performed by the low-pass filter. Note that we multiply the envelope signal several times to be able to compare its fit to the original signal.

In [129]:
```
# plot rectified EMG signal with superimposed envelope detection with lo
w-pass filter
plt.figure(figsize=(18,6))
plt.xlabel('time (s)')
plt.ylabel('voltage ($\mu$V)')
plt.plot(timeAbs,absSignal, 'b', alpha=0.4)
plt.plot(timeAbs,datafrle*3, 'r', linewidth=2.5) # multiply envelope to
 see data fit
plt.xlim(0,max(xx['timeEMG']))
plt.ylim(-400,7500);
```



Study questions and exercises:

- Describe the effects of low-pass filtering the EMG signal.
- Why did we specify 8 Hz as the low-pass cutoff?
- What would happen if you changed the cutoff? Experiment with different cutoffs. Graph and compare the results.
- Are there other techniques you could use to calculate the envelope of the rectified EMG signal? Describe them and their advantages/disadvantages. Apply at least one of these techniques and graph the results.

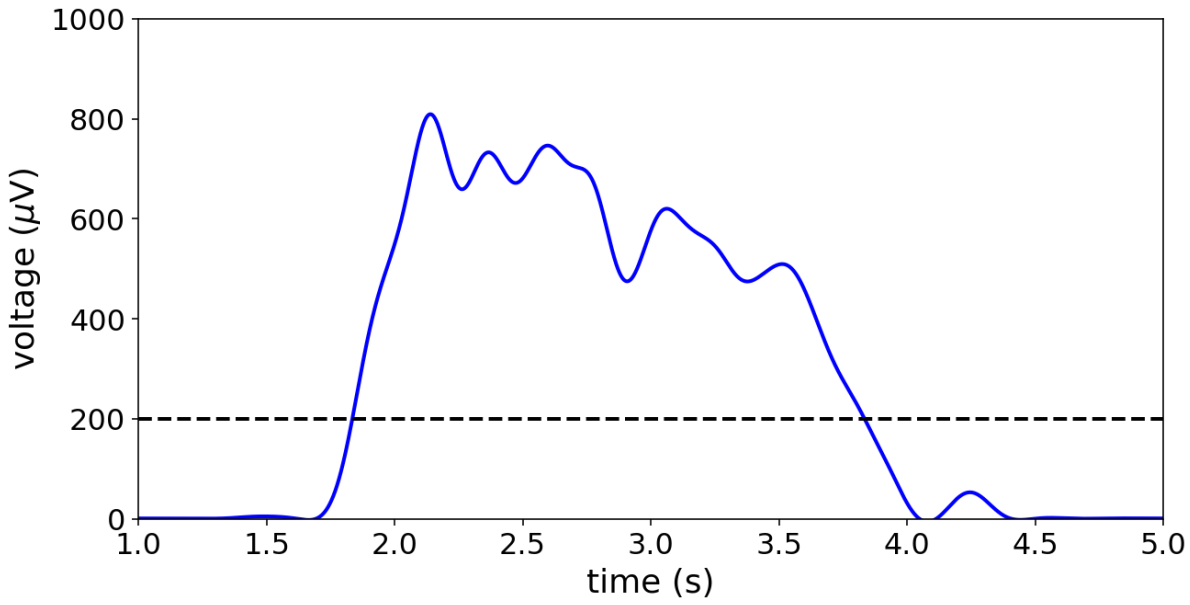
Analyzing the data

When analyzing muscle activity, it is useful to know the start and stop times of the contractions. With this information, for example, we can quantify the contraction durations and see how much these vary over a recording. In other words, is the subject able to maintain contractions of the same duration, or do the contractions get shorter over the course of the recording? Or, we can quantify how long a subject is able to maintain a sustained contraction, and then quantify how this changes over repeated trials. Both of these quantifications could allow us to look at how particular exercises induce muscle fatigue.

Detecting start and stop times of muscle contraction

If we plot just the EMG envelope (not the multiplied signal superimposed on the rectified EMG), we can then estimate the best value to use as a threshold to detect muscle contraction start and stop times. We want a value low enough to include the majority of the contraction, but large enough to exclude noise and small muscle twitches. For the envelope plotted below, it looks like 200 (dashed line) is a good candidate threshold.

In [136]:
```
# plot just the envelope to determine threshold for detecting contractio
ns
plt.figure(figsize=(10,5))
plt.xlabel('time (s)')
plt.ylabel('voltage ($\mu$V)')
plt.plot(timeAbs,datafrle, 'b', linewidth=2.0)
plt.axhline(y=200, color='k',linewidth=2.0, linestyle='--') # potential
 threshold
plt.xlim(1,5)
plt.ylim(0,1000);
```



We can then use our threshold to detect the contraction start and stop times. For the start times, we want the indices where the signal value is equal to or greater than the threshold when the previous value was below the threshold, i.e. the ascending threshold crossing.

In [133]:
```
threshold = 200
cstarts = []
for i in range(1, len(datafrle)):
    if datafrle[i-1] < threshold and datafrle[i] >= threshold:
        cstarts.append(i)

print(cstarts)
```

```
[80940, 326408, 556036, 797722]
```

For the stop times, we want the indices where the signal value is below the threshold when the previous value was equal to or above the threshold, i.e. the descending threshold crossing.

In [134]:
```
threshold = 200
cstops = []
for i in range(1, len(datafrle)):
    if datafrle[i-1] >= threshold and datafrle[i] < threshold:
        cstops.append(i)

print(cstops)
```

```
[169099, 425989, 654532, 897724]
```

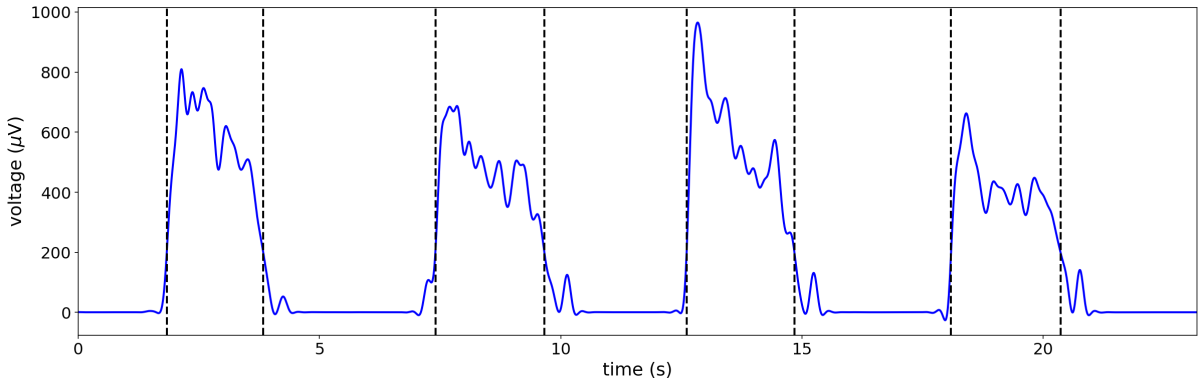
We then divide the start and stop times by the sampling rate to get the times in seconds.

In [135]:
```
starts=[]
for n in range(0, len(cstarts)):
    starttime = cstarts[n]/xx['sampleRate']
    starts.append(starttime)

print(starts)

stops=[]
for n in range(0, len(cstops)):
    stoptime = cstops[n]/xx['sampleRate']
    stops.append(stoptime)

print(stops)
```

```
[1.835374149659864, 7.401541950113379, 12.608526077097507, 18.088934240
36281]
[3.8344444444444443, 9.659614512471656, 14.841995464852607, 20.35655328
798186]
```

To see how well our event detection works, we plot the enveloped EMG signal and then plot the start and stop times with dashed lines. We can go back and adjust the threshold depending on whether we want to capture more or less of the muscle activity.

In [137]:
```
# plot envelope with contraction start and stop times marked by dashed l
ines
plt.figure(figsize=(20,6))
plt.xlabel('time (s)')
plt.ylabel('voltage ($\mu$V)')
plt.plot(timeAbs, datafrle, 'b', linewidth=2.0)
for n in range(0, len(starts)):
    plt.axvline(x=starts[n], color='k',linewidth=2.0, linestyle='--')
for n in range(0, len(stops)):
    plt.axvline(x=stops[n], color='k',linewidth=2.0, linestyle='--')
plt.xlim(0, max(xx['timeEMG']));
```



We can see from the above figure that our contraction detection works well. With the selected threshold, we are detecting the start and stop times of the principle contractions, while excluding the small muscle twitches following each. We can adjust the threshold if needed to improve the detection. Remember also that the threshold will have to be determined for each recording.

Now, by taking the difference between the successive start and stop times, we can calculate the duration of each muscle contraction. We print the durations rounded up to two decimal places. Units are in seconds.

In [138]:
```
# calculate and print the contraction durations in seconds
durs=[]
for n in range(0,len(starts)):
    durs.append(stops[n]-starts[n])
    print(round(durs[n], 2))
```

```
2.0
2.26
2.23
2.27
```

Study questions and exercises:

- Decrease/increase the threshold, plot the results, and describe what happens to contraction detection.
- What are the disadvantages of the technique we used to detect contraction start and stop times?
- Are there other techniques you could use to detect the start and stop times? If so, what are they? How would you implement them?
- Write and implement code that will allow you to quantify other aspects of the muscle activity, such as the rest interval between each contraction.

Analyzing more recordings

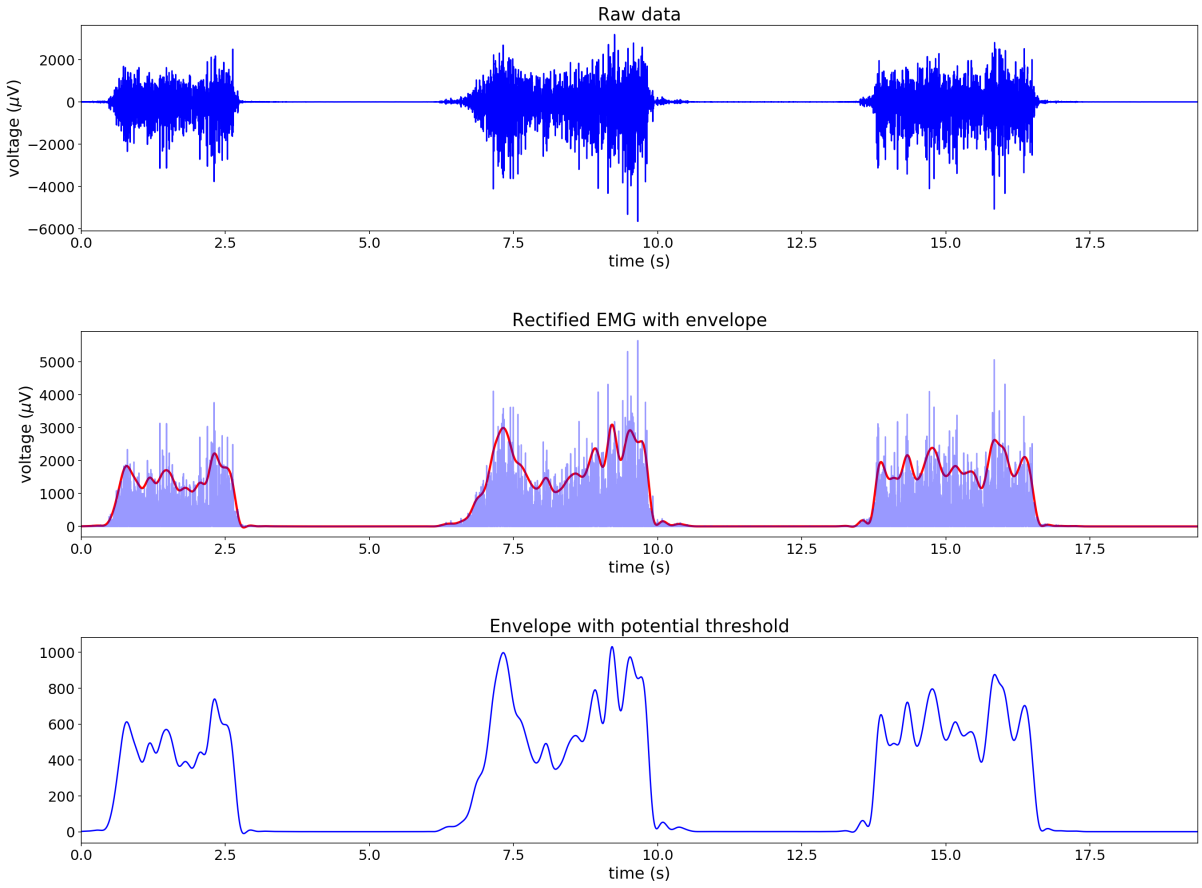
To analyze more recordings, we first define a function that will allow us to quickly run the basic data extraction, visualization, and signal processing. This way we can easily look at different recordings just by changing the file name, or even batch run multiple files using a simple for loop. This function includes many of the same commands we had in our previous function, but now we add in the filtering, signal rectification, and envelope detection. An additional input to the function also allows us to specify whether we want to graph the data or just run the function without graphing.

In [239]:
```
def processEMG(file,graph):
    # open .wav file by specifying the path and filename
    record = wave.open(file)
    # extract number of channels, sample rate, data
    numChannels = record.getnchannels() # number of channels
    N = record.getnframes() # humber of frames
    sampleRate = record.getframerate() # sampling rate
    # extract data from the .wav file
    dstr = record.readframes(N * numChannels)
    waveData = np.frombuffer(dstr, np.int16)
    # calculate time window
    timeEMG=np.linspace(0, len(waveData)/sampleRate, num=len(waveData))
    # calculate frequency
    freq = 1/np.mean(np.diff(timeEMG))
    # band-pass Butterworth filter at 10-400Hz
    b, a = butter(2, ([10, 400]/(freq/2)), btype = 'bandpass')
    dataf = filtfilt(b, a, waveData)
    # rectify filtered EMG signal
    absSignal = np.absolute(dataf)
    timeAbs=np.linspace(0, len(absSignal)/sampleRate, num=len(absSignal
))
    # low-pass Butterworth filter for envelope detection
    lowp = 8
    sfreq = sampleRate
    low_pass = lowp/sfreq
    b, a = sc.signal.butter(4, low_pass, btype='lowpass')
    datafrle = filtfilt(b, a, absSignal)
    # save relevant data in array
    xx={'sampleRate':sampleRate,\
        'waveData':waveData,\
        'timeEMG':timeEMG,\
        'freq':freq,\
        'dataf':dataf,\
        'absSignal':absSignal,\
        'timeAbs':timeAbs,\
        'datafrle':datafrle
        }
    # plot EMG
    if graph:

        fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize = (20,15))

        # plot filtered data
        ax1.plot(timeEMG,dataf, 'b')
        ax1.set_title('Raw data', fontsize=20)
        ax1.set_xlim(0,max(timeEMG))
        #ax1.set_ylim(-8000,4000)
        ax1.set_xlabel('time (s)')
        ax1.set_ylabel('voltage ($\mu$V)')

        # plot rectified data with envelope
        ax2.plot(timeAbs,datafrle*3, 'r', linewidth=2.5) # multiply enve
lope to see data fit
        ax2.plot(timeAbs,absSignal, 'b', alpha=0.4)
        ax2.set_title('Rectified EMG with envelope', fontsize=20)
        ax2.set_xlim(0,max(timeEMG))
```

```
        ax2.set_xlabel('time (s)')
        ax2.set_ylabel('voltage ($\mu$V)')

        # plot envelope with potential threshold
        ax3.plot(timeAbs,datafrle, 'b')
        #plt.axhline(y=200, color='k',linewidth=2.0, linestyle='--') # p
otential threshold
        ax3.set_title('Envelope with potential threshold', fontsize=20)
        ax3.set_xlim(0,max(timeEMG))
        #ax3.set_ylim(0,1200)
        ax3.set_xlabel('time (s)')

        # spacing between subplots
        plt.tight_layout(pad=3.0)

    return xx
```

Now we can run the function on a different file. The following EMG was recorded from the bicep muscle during repeated contractions.

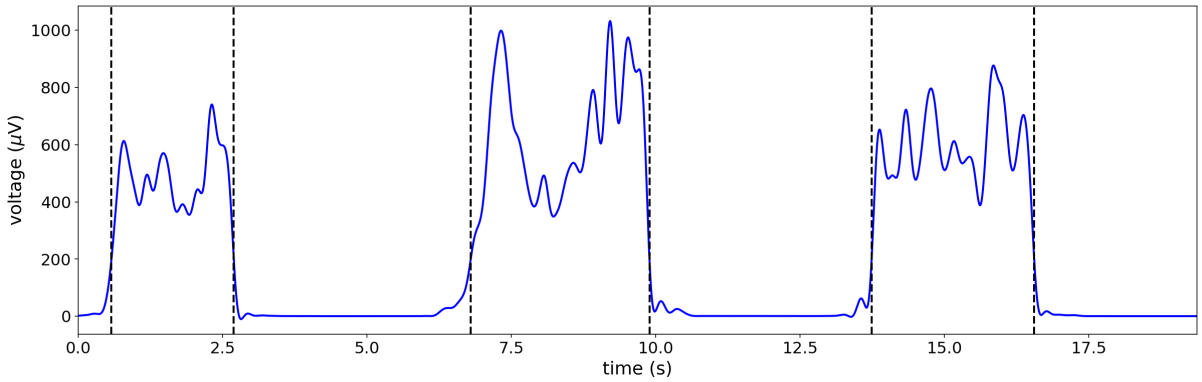
In [240]:
```
xx=processEMG(file='../data/S10_EMG_bicep_intermittent.wav',graph=1)
```



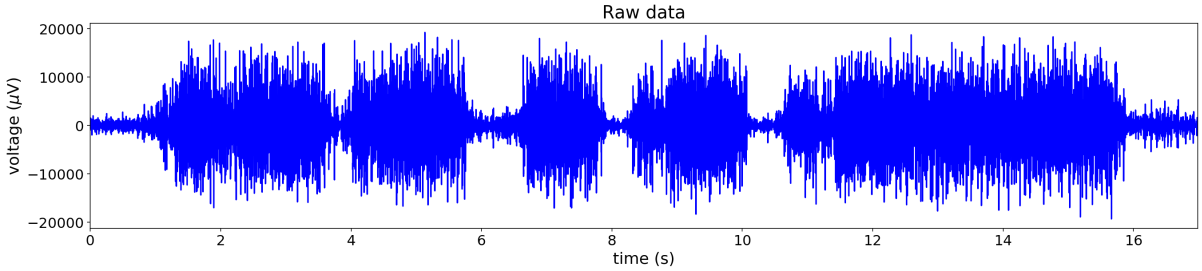
In [241]:
```
def contractions(threshold):

    # ascending threshold crossings
    cstarts = []
    for i in range(1, len(xx['datafrle'])):
        if xx['datafrle'][i-1] < threshold and xx['datafrle'][i] >= thre
shold:
            cstarts.append(i)

    # descending threshold crossings
    cstops = []
    for i in range(1, len(xx['datafrle'])):
        if xx['datafrle'][i-1] >= threshold and xx['datafrle'][i] < thre
shold:
            cstops.append(i)

    # divide start and stop times by the sampling rate to get times in s
econds
    starts=[]
    for n in range(0, len(cstarts)):
        starttime = cstarts[n]/xx['sampleRate']
        starts.append(starttime)

    stops=[]
    for n in range(0, len(cstops)):
        stoptime = cstops[n]/xx['sampleRate']
        stops.append(stoptime)

    # plot envelope with contraction start and stop times marked by dash
ed lines
    plt.figure(figsize=(20,6))
    plt.xlabel('time (s)')
    plt.ylabel('voltage ($\mu$V)')
    plt.plot(xx['timeAbs'], xx['datafrle'], 'b', linewidth=2.0)
    for n in range(0, len(starts)):
        plt.axvline(x=starts[n], color='k',linewidth=2.0, linestyle='--'
)
    for n in range(0, len(stops)):
        plt.axvline(x=stops[n], color='k',linewidth=2.0, linestyle='--')
    plt.xlim(0, max(xx['timeEMG']))

    # calculate contraction durations
    durs=[]
    for n in range(0,len(starts)):
        durs.append(stops[n]-starts[n])
        print('Contraction lasted %g seconds'%(round(durs[n], 2)))

    return
```

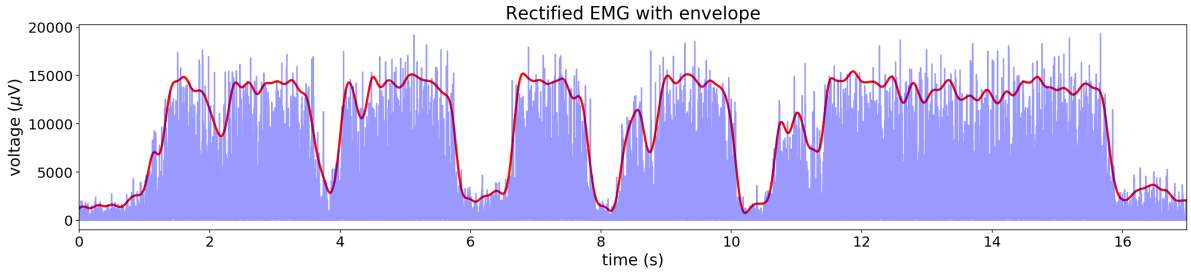
In [242]:
```
contractions(threshold=200)
```

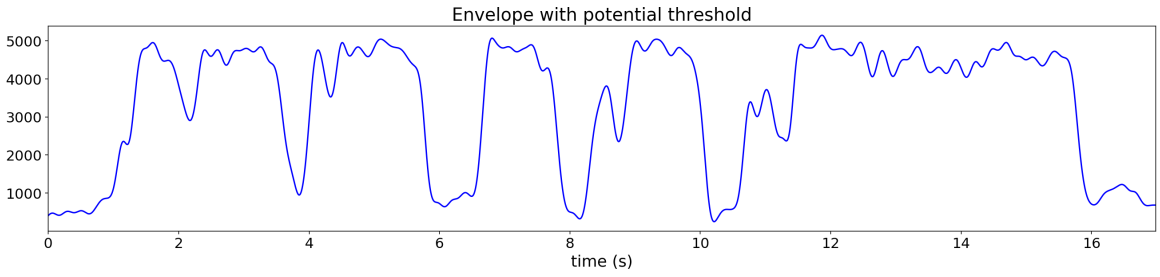
```
Contraction lasted 2.12 seconds
Contraction lasted 3.09 seconds
Contraction lasted 2.81 seconds
```



In [243]:
```
xx=processEMG(file='../data/S2_EMG_forearm_grip.wav',graph=1)
```







In [245]:
```
contractions(threshold=1400)
```

```
Contraction lasted 2.72 seconds
Contraction lasted 1.92 seconds
Contraction lasted 1.29 seconds
Contraction lasted 1.81 seconds
Contraction lasted 5.25 seconds
```