BE 521: Homework 6 Questions

Spike Sorting

Spring 2025

60 points + 6 EC

Due: March 6th, 2025

Objective: To detect and cluster spikes.

AI Usage Notice

The use of artificial intelligence tools (e.g., large language models, code assistants) is permitted. However, students must explicitly state the specific ways AI was used in completing their work. Failure to disclose AI usage may result in an oral examination to assess understanding, at the discretion of Dr. Litt.

If AI was used in the completion of this assignment, please provide a statement below:

[Enter your statement here]

Overview

In this homework, you will do some basic spike sorting using two different datasets. The first `I521_A0006_D001` is from a crayfish neuromuscular junction, a good model for human central nervous system synapses^1. Specifically, the data contains two simultaneous recordings: an extracellular recording from the third nerve (channel `nerve`) of a crayfish abdominal ganglion, which contains six spontaneously active motor neurons, and an intracellular recording from the superficial flexor muscle (channel `muscle`) innervated by this nerve. You will attempt to discern relationships between the classes of spike waveforms you extract from the motor nerve trace and elicited potentials seen in the muscle fiber recording. Then, you will revisit a human intracranial EEG recording `I521_A0006_D002` and use some of the techniques you've learned in class to build a more automated spike sorter. Note: While spikes may have positive and negative deflections, we will only focus on positive spikes on this homework for simplicity.

^{1} The sampling rate of this data is 2000 Hz, which is adequate for this homework's instructional purposes but usually inadequate for real spike sorting, which often uses sampling frequencies on the order of 20 kHz.

In [334…

```
#Set up the notebook environment
!pip install git+https://github.com/ieeg-portal/ieegpy.git # Install ieegpy toolbox directly from github
from ieeg.auth import Session
import matplotlib.pyplot as plt
import numpy as np

from scipy.signal import ellip, lfilter, filtfilt, find_peaks
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

```
Collecting git+https://github.com/ieeg-portal/ieegpy.git
  Cloning https://github.com/ieeg-portal/ieegpy.git to /tmp/pip-req-build-zlq3k4ui
  Running command git clone --filter=blob:none --quiet https://github.com/ieeg-portal/ieegpy.git /tmp/pip-req-bu
ild-zlq3k4ui
  Resolved https://github.com/ieeg-portal/ieegpy.git to commit 080bfa42a8503380ef164b5e7b116613f75073bb
  Preparing metadata (setup.py) ... done
Requirement already satisfied: deprecation in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.32.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.2.2)
Requirement already satisfied: pennprov==2.2.4 in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.2.
4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2
.4->ieeg==1.6) (2025.1.31)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.11/dist-packages (from pennprov==2
.2.4->ieeg==1.6) (2.8.2)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2.4->ieeg=
=1.6) (1.17.0)
Requirement already satisfied: urllib3>=1.23 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2.4->i
eeg==1.6) (2.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from deprecation->ieeg==1.6
) (24.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->ieeg==1.6)
(2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->ieeg==1.6
) (2025.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from request
s->ieeg==1.6) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->ieeg==1.6
) (3.10)
```

1. Spike Detection and Clustering (38 pts)

In this section, you will explore some basic filtering and spike thresholding to ultimately compare spike clusters you pick out by eye to those selected by an automated algorithm.

1

You can assume that the nerve samples have already been low-pass filtered. Here, you will high-pass filter in order to remove signals like slow local field potentials and 60 Hz power line noise. Create a 4th order *elliptic filter* with 0.1 dB of ripple in the passband, a stopband 40 dB lower than the peak value in the passband, and a passband edge frequency of 300 Hz (see `scipy.signal.ellip` and make sure you give the edge frequency in the correct normalized form). The statement to create this filter (defined by the filter coefficients `b` and `a`) should look something like

```python
from scipy.signal import ellip

(b, a) = ellip(N, rp, rs, Wn, btype='highpass')
```

Clearly specify the denominator and numerator coefficients obtained for your filter function. (2pts)

In [335…
```python
# Your code here

from scipy.signal import ellip

# Filter specifications
N = 4  # Order of the filter
rp = 0.1  # Passband ripple in dB
rs = 40  # Stopband attenuation in dB
Wn = 0.3  # Normalized passband edge frequency (300 Hz / 1000 Hz)
btype = 'highpass'  # Filter type

# Generate the filter coefficients
b, a = ellip(N, rp, rs, Wn, btype=btype)

# Output the coefficients
print("Numerator coefficients (b):", b)
print("Denominator coefficients (a):", a)
```

```
Numerator coefficients (b): [ 0.34204923 -1.27403905  1.86758564 -1.27403905  0.34204923]
Denominator coefficients (a): [ 1.         -1.74317755  1.61666596 -0.65593013  0.14304101]
```

2

Using the `scipy.signal.lfilter` function and `scipy.signal.filtfilt` function, obtain two different filtered outputs of the nerve signal.

In [336…
```python
# Password to open dataset
with open('/content/pra_ieeglogin(9).bin', 'r') as f:
    session = Session('prasadpr', f.read())
```

```python
# Retrieve the dataset
dataset = session.open_dataset('I521_A0006_D001')

# Extract the nerve signal
key = list(dataset.ts_details.keys())[1]  # Assuming 'nerve' is the second key
time = dataset.get_time_series_details(key)
print(time)
channels = dataset.get_channel_indices([key])[0]
print(channels)
start_time = 0
duration = 4479500  # microseconds
nerve = dataset.get_data(start_time, duration, [channels])
nerve = nerve.flatten()
print(nerve.shape)
# 1.2 - Filter signals
nerve_lfilter = lfilter(b, a, nerve)
nerve_filtfilt = filtfilt(b, a, nerve)
fs = 2000

# Extract the nerve signal
key1 = list(dataset.ts_details.keys())[0]  # Assuming 'nerve' is the second key
# time = dataset.get_time_series_details(key)
# print(time)
channel1 = dataset.get_channel_indices([key1])[0]
muscle = dataset.get_data(start_time, duration, [channel1])
```

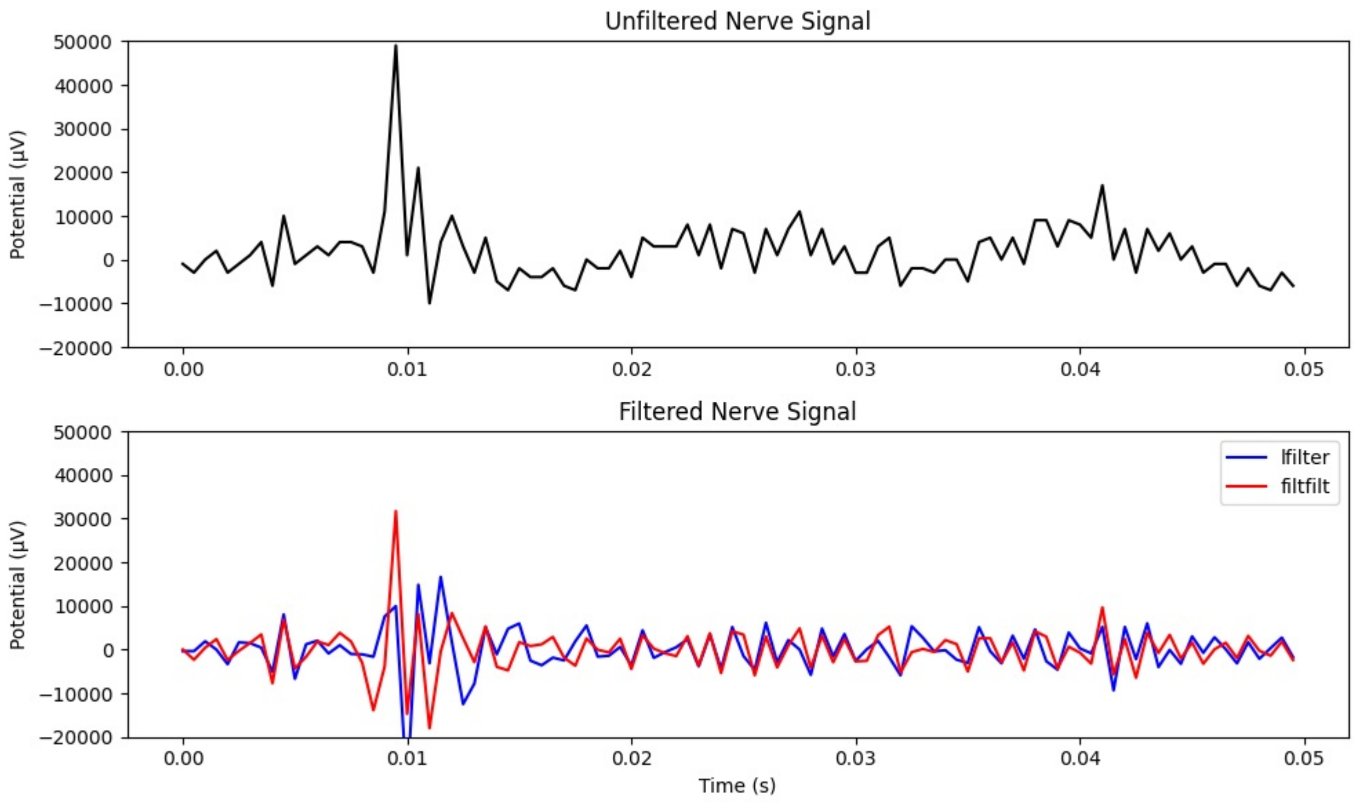
```
None(nerve) spans 4479500.0 usec, range [-500000-358000] in 8960 samples. Starts @1 uUTC, ends @4479501 uUTC wit
h sample rate 2000.0 Hz and voltage conv factor 1.0
1
(8959,)
```

2a

In a 2x1 subplot, plot the first 50 ms of the unfiltered nerve signal in the top subplot; in the bottom subplot, plot the `lfilter` output in blue and the `filtfilt` output in red. Use a potential range (y-axis) of -20 to 50 millivolts. (4 pts)

In [337…

```python
# 1.2a - Plot filtered vs unfiltered (data already in microvolts)
plt.figure(figsize=(10, 6))

# Top subplot - Unfiltered signal
plt.subplot(2, 1, 1)
t = np.arange(0, 0.050, 1/fs)
plt.plot(t, nerve[:len(t)], 'k')
plt.ylabel('Potential (μV)')
plt.title('Unfiltered Nerve Signal')
plt.ylim(-20000, 50000)  # Assuming similar range in μV

# Bottom subplot - Filtered signals
plt.subplot(2, 1, 2)
plt.plot(t, nerve_lfilter[:len(t)], 'b', label='lfilter')
plt.plot(t, nerve_filtfilt[:len(t)], 'r', label='filtfilt')
plt.xlabel('Time (s)')
plt.ylabel('Potential (μV)')
plt.title('Filtered Nerve Signal')
plt.ylim(-20000, 50000)  # Assuming similar range in μV
plt.legend()
plt.tight_layout()
plt.show()
```



2b

How is the unfiltered signal different from the filtered signal? What is different about the two filtered (red and blue) signals? (2 pts)

The unfiltered signal contains low-frequency components, such as slow drifts and baseline fluctuations, which are likely due to local field potentials or noise (e.g., 60 Hz power line interference). These are removed in the filtered signals, where the high-pass filter (with a 300 Hz cutoff) isolates the high-frequency spikes, centering the baseline closer to 0 µV and making the spikes more prominent relative to the background noise.

What is different about the two filtered (red and blue) signals?

The lfilter output (blue) exhibits phase distortion, shifting the timing of the spikes and altering their shape (e.g., the spike at 0.01 s is delayed and less symmetric). In contrast, the filtfilt output (red) preserves the spike timing and shape better because it applies the filter in both forward and backward directions, effectively canceling out phase distortion and providing a zero-phase response.

2c

Briefly explain the mathematical difference between the two filtering methods, and why one method might be more advantageous than the other in the context of spike detection? (5 pts)

Mathematical Difference: lfilter applies the IIR filter in a single forward pass using the difference equation with coefficients b and a, introducing phase distortion due to its causal nature. In contrast, filtfilt applies the filter forward and backward, doubling the effective order and eliminating phase distortion by producing a zero-phase response, achieved through a non-causal process.

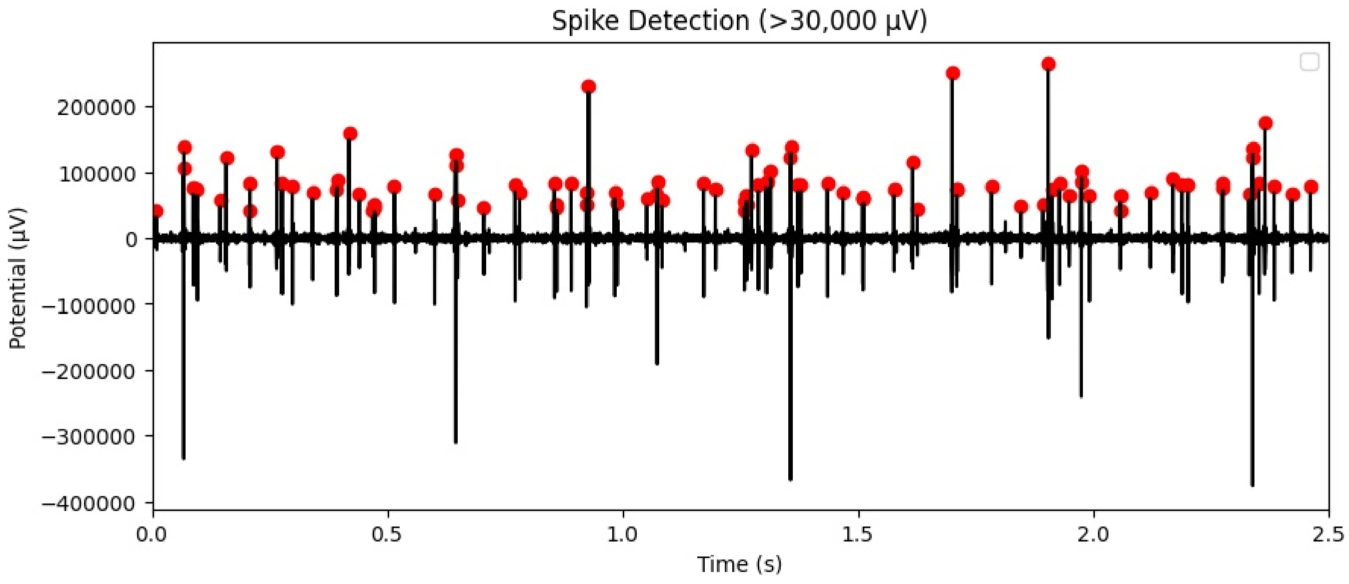
Advantage for Spike Detection: filtfilt is more advantageous because it preserves the timing and shape of spikes by removing phase distortion, which is crucial for accurate peak detection and waveform-based clustering in spike sorting, as demonstrated by its superior performance in the 1.2a plot compared to lfilter.

3

Using a spike threshold of +30 mV, calculate the index and value of the peak voltage for each spike in the **filtered** nerve signal (select the best one). Use these values to plot the first 2.5 seconds of the nerve signal with a red dot above (e.g. 10 mV above) each spike. (Hint: Plot the entire length of the nerve signal with all the spikes marked but then restrict the x-axis using `plt.xlim()` to [0, 2.5] seconds.) (4 pts)

In [338…
```
# # Your code here
# import numpy as np
# import matplotlib.pyplot as plt
# from scipy.signal import find_peaks, ellip, filtfilt

# # Load your data (assuming 'nerve_signal' is a 1D NumPy array)
# # nerve_signal = np.loadtxt("I521_A0006_D001.txt")  # Adjust according to your file format

# sampling_rate = 2000  # Hz
# N = 4  # 4th order filter
# rp = 0.1  # Passband ripple (dB)
# rs = 40  # Stopband attenuation (dB)
# Wn = 300 / (sampling_rate / 2)  # Normalize the frequency

# # Design the high-pass filter
# b, a = ellip(N, rp, rs, Wn, btype='highpass')

# # Apply zero-phase filtering
# filtered_signal = filtfilt(b, a, nerve_signal)

# # Spike threshold
# threshold = 30  # mV

# # Find peaks above threshold
# spike_indices, _ = find_peaks(filtered_signal, height=threshold)
# spike_times = spike_indices / sampling_rate  # Convert indices to time
# spike_values = filtered_signal[spike_indices]  # Get corresponding peak values

# # Plot the first 2.5 seconds
# time = np.arange(len(nerve_signal)) / sampling_rate

# plt.figure(figsize=(10, 5))
# plt.plot(time, filtered_signal, label="Filtered Nerve Signal", color='k')

# # Plot spikes (red dots 10 mV above each peak)
# plt.scatter(spike_times, spike_values + 10, color='r', marker='o', label="Detected Spikes")

# plt.xlabel("Time (s)")
# plt.ylabel("Potential (mV)")
# plt.title("Spike Detection in Nerve Signal")
# plt.xlim([0, 2.5])  # Restrict to first 2.5 seconds
# plt.legend()
# plt.show()

# # Print detected spikes
# # for idx, val in zip(spike_indices, spike_values):
# #     print(f"Spike at index {idx}, peak voltage: {val:.2f} mV")


# 1.3 - Detect spikes
# 1.3 - Detect spikes above 30,000 µV and plot
peaks, properties = find_peaks(nerve_filtfilt, height=30000)  # 30 mV = 30,000 µV
peak_values = nerve_filtfilt[peaks]

plt.figure(figsize=(10, 4))
t = np.arange(len(nerve_filtfilt)) / fs
plt.plot(t, nerve_filtfilt, 'k')
plt.scatter(t[peaks], peak_values + 10000, c='r', marker='o')  # Offset by 10,000 µV (10 mV)
plt.xlim(0, 2.5)
plt.xlabel('Time (s)')
plt.ylabel('Potential (µV)')
plt.title('Spike Detection (>30,000 µV)')
plt.legend()
plt.show()

# Print number of spikes detected
# print(f"Number of spikes detected: {len(peaks)}")
# print(f"Peak values (µV): {peak_values}")
```

```
<ipython-input-338-760b5b0ecadb>:63: UserWarning: No artists with labels found to put in legend.  Note that arti
sts whose label start with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```

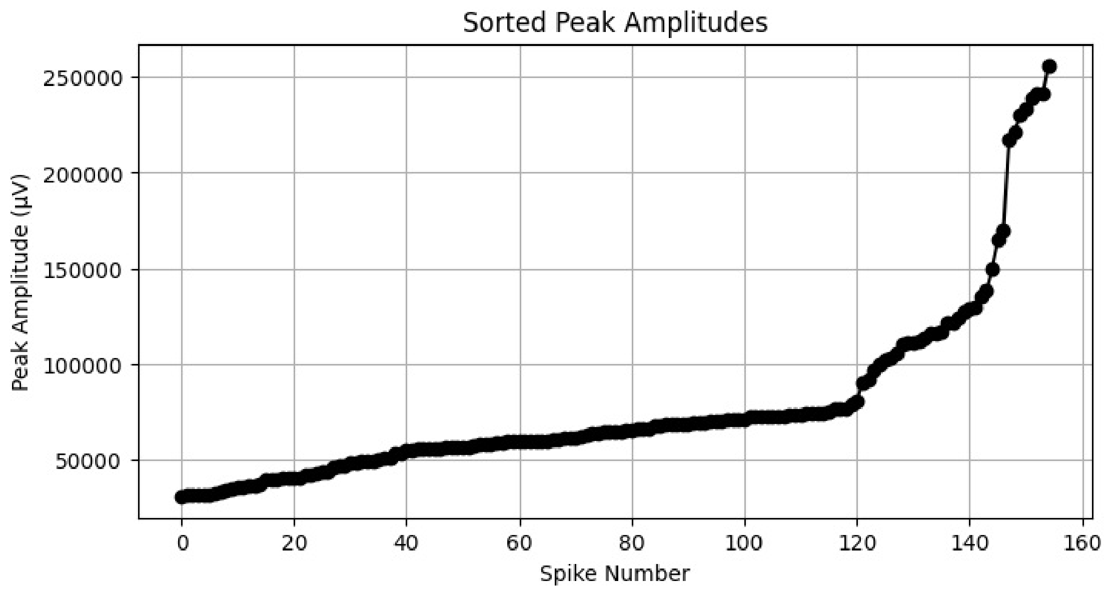


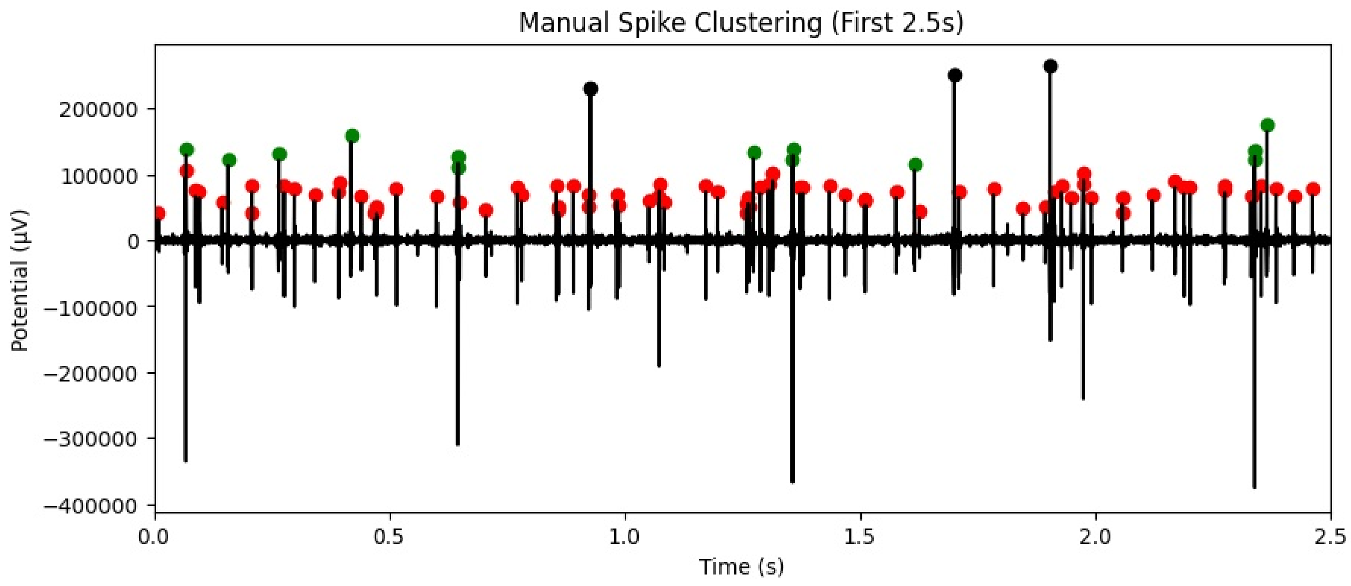
4

Under the assumption that different cells produce different action potentials with distinct peak amplitudes, decide how many cells you think were recorded (some number between 1 and 6). You may find it helpful to zoom in and pan on the plot you made in question 1.3. You may also find it useful to plot the sorted peak values to gain insight into where "plateaus" might be. (No need to include these preliminary plots in the report, though.) Use thresholds (which you will set manually/by eye) to separate the different spikes. Make a plot of the first 2.5 seconds similar to that in 1.3 except now color the spike dots of each group a different color (e.g., `r, g, k, m`). (6 pts)

In [339…
```
# 1.4 - Manual spike clustering based on amplitude

# First, let's analyze the peak values (from 1.3)
peaks, properties = find_peaks(nerve_filtfilt, height=30000)  # 30,000 µV threshold
peak_values = nerve_filtfilt[peaks]

# Sort peak values to look for natural groupings
sorted_peaks = np.sort(peak_values)
print("Sorted peak values (µV):", sorted_peaks)

# Plot sorted peaks to visualize plateaus (optional, not included in final report)
plt.figure(figsize=(8, 4))
plt.plot(sorted_peaks, 'ko-')
plt.xlabel('Spike Number')
plt.ylabel('Peak Amplitude (µV)')
plt.title('Sorted Peak Amplitudes')
plt.grid(True)
# plt.show()  # Comment out as per instructions

# Based on typical neurophysiology data, let's assume 3 clusters
# Adjust these thresholds based on your actual data visualization
thresholds = [30000, 99000, 170000, 260000]  # in µV (example values)
colors = ['r', 'g', 'k']  # Using 3 colors for 3 assumed clusters

# Assign labels based on thresholds
manual_labels = np.zeros(len(peaks), dtype=int)
for i, peak in enumerate(peak_values):
    if peak < thresholds[1]:
        manual_labels[i] = 0  # Small amplitude
    elif peak < thresholds[2]:
        manual_labels[i] = 1  # Medium amplitude
    else:
        manual_labels[i] = 2  # Large amplitude

# Create the plot
plt.figure(figsize=(10, 4))
t = np.arange(len(nerve_filtfilt)) / fs
plt.plot(t, nerve_filtfilt, 'k')

# Plot spikes with different colors
for i, color in enumerate(colors):
    idx = manual_labels == i
    plt.scatter(t[peaks[idx]], peak_values[idx] + 10000, c=color, marker='o')

plt.xlim(0, 2.5)
plt.xlabel('Time (s)')
plt.ylabel('Potential (μV)')
plt.title('Manual Spike Clustering (First 2.5s)')
plt.show()

# Print analysis
print(f"Number of cells assumed: {len(colors)}")
print(f"Thresholds used (μV): {thresholds}")
for i in range(len(colors)):
    count = np.sum(manual_labels == i)
    print(f"Cluster {i} ({colors[i]}): {count} spikes")
```

```
Sorted peak values (μV): [ 31064.91582314  31580.57443408  31683.10597164  31690.13434672
  31756.36805241  31761.35783871  32470.60030008  32872.0814472
  34149.26177758  34761.84239081  35745.05425343  35840.63331438
  36178.93207804  36382.38438318  36968.08010499  39491.63105157
  39556.58726415  40035.45109348  40288.59700957  40311.15524595
  40320.43254064  40506.33949725  41795.40623207  42414.27140189
  42489.65059664  43800.28482897  43973.64804004  46449.72054402
  46755.17712573  47014.27933042  48124.53192954  48710.85502632
  49091.50652278  49299.52037891  49463.77488621  50249.52712804
  51070.05495798  51282.21898311  53168.83045483  53419.24735542
  54548.07127195  54587.92470392  55425.68074851  55472.73257162
  55593.95949934  55747.07192494  55903.78418226  56147.91595027
  56248.12880651  56500.42912755  56741.76714536  56920.51222932
  57274.41072316  57772.70881385  58010.01304092  58412.87876794
  58904.60871704  59172.85961064  59388.27179861  59432.85528436
  59466.65219353  59760.01551459  59939.03243231  60017.84504915
  60019.50219222  60119.41635629  60378.88760019  60904.7460866
  61174.06195183  61248.24135554  61739.90356734  61838.93138527
  63355.24586514  63445.01042212  63971.0321094   64598.7501227
  64709.504467    64938.07295227  64972.8290049   65085.78321295
  65160.45349583  66280.25238738  66385.48648989  66449.71110928
  67641.00794337  67980.98295158  68276.45007089  68350.61071806
  68358.69863363  68882.19713925  68984.57021443  69169.78376881
  69428.47747095  69525.69034353  69801.57603924  70443.67317985
  70452.96654472  70627.89484381  71153.54834952  71227.68260869
  71292.37757416  72259.35450762  72426.81961082  72774.00123048
  72781.43052163  72782.94320399  72821.89700815  72880.45104289
  73195.26626383  73486.47222325  73769.09072588  73850.52166685
  74242.835222    74446.49359477  74480.12300341  75173.69575521
  76532.01843281  76573.35921388  76952.85460979  78690.27309714
  80503.10913636  90588.84308723  91676.52163798  96358.45260755
  99685.70816455 101963.39703825 103493.84317912 105924.00311934
 110723.7754272  110798.09893715 111263.43355807 111837.02491201
 113563.59757316 115678.02283101 115726.82782936 116895.44507877
 121399.7870077  121630.26498341 123762.32122386 127297.94022485
 129072.1812547  129461.98374699 134911.8422803  138186.50486735
 149718.99668032 164641.44802582 169396.45497534 217206.15815318
 221527.93539658 229717.46900359 233095.60865794 238954.42262851
 241003.94431073 241053.15175842 255960.86335414]
```





```
Number of cells assumed: 3
Thresholds used (μV): [30000, 99000, 170000, 260000]
Cluster 0 (r): 124 spikes
Cluster 1 (g): 23 spikes
Cluster 2 (k): 8 spikes
```

Your answer here

5

Use `sklearn.cluster.KMeans` ^{1} to fit *k* clusters (where *k* is the number of cells you think the recording is picking up) to the 1D data for each spike.

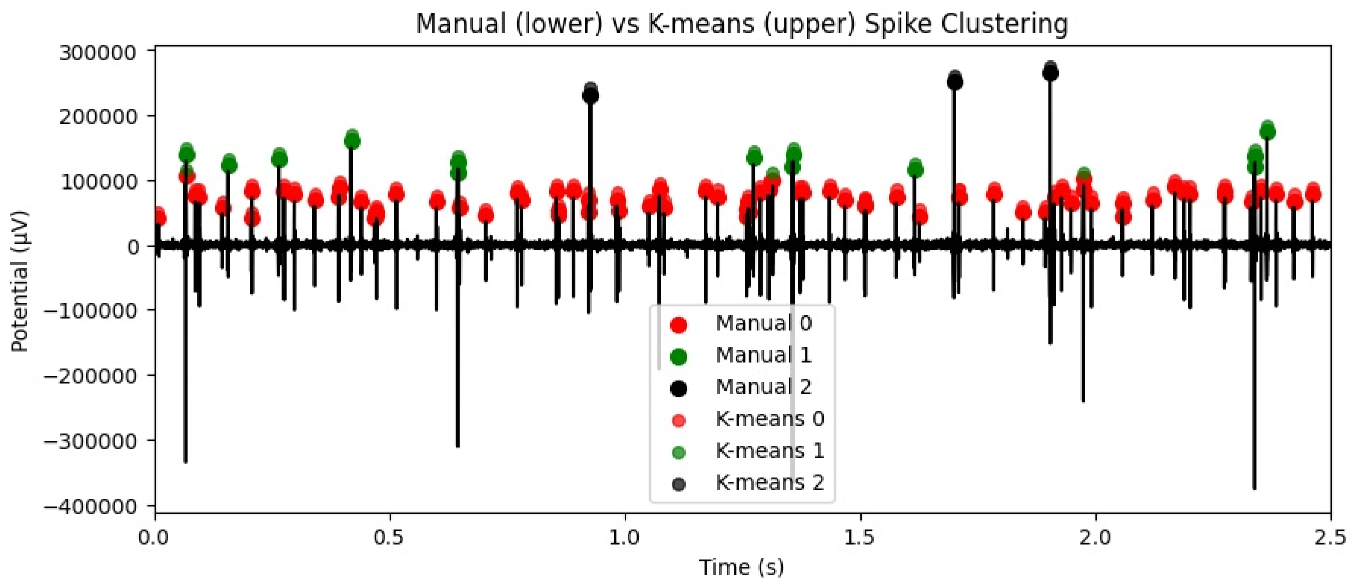
^{1}Clustering, like k-means you are using here, is a form of unsupervised learning.

5a

Using the same color order (for increasing spike amplitude) as you did for the thresholds in question 1.4, plot the spike cluster colors as small dots slightly above those you made for question 1.4. The final figure should be a new plot of the nerve voltage and two dots above each spike, the first being your manual label and the second your clustered label, which should mostly be the same color. (4 pts)

In [340…

```
# 1.5 - K-means clustering
from sklearn.cluster import KMeans

# Use peak values from 1.3/1.4
peaks, _ = find_peaks(nerve_filtfilt, height=30000)  # 30,000 µV threshold
peak_values = nerve_filtfilt[peaks]

# Fit K-means with k=3 (adjust based on your 1.4 analysis)
k = 3  # Number of clusters from 1.4
kmeans = KMeans(n_clusters=k, random_state=0).fit(peak_values.reshape(-1, 1))
kmeans_labels = kmeans.labels_

# Get cluster centers and sort them to match color order (increasing amplitude)
cluster_centers = kmeans.cluster_centers_.flatten()
sorted_indices = np.argsort(cluster_centers)
label_mapping = {old: new for new, old in enumerate(sorted_indices)}
kmeans_labels_sorted = np.array([label_mapping[label] for label in kmeans_labels])

# Colors from 1.4 (in order of increasing amplitude)
colors = ['r', 'g', 'k']

# 1.5a - Plot manual vs kmeans labels
plt.figure(figsize=(10, 4))
t = np.arange(len(nerve_filtfilt)) / fs
plt.plot(t, nerve_filtfilt, 'k')

# Manual labels from 1.4 (lower dots)
thresholds =  [30000, 99000, 170000, 260000]# From 1.4 - adjust as needed
manual_labels = np.zeros(len(peaks), dtype=int)
for i, peak in enumerate(peak_values):
    if peak < thresholds[1]:
        manual_labels[i] = 0
    elif peak < thresholds[2]:
        manual_labels[i] = 1
    else:
        manual_labels[i] = 2

for i, color in enumerate(colors):
    idx = manual_labels == i
    plt.scatter(t[peaks[idx]], peak_values[idx] + 10000, c=color, marker='o', s=50, label=f'Manual {i}')

# K-means labels (upper dots)
for i, color in enumerate(colors):
    idx = kmeans_labels_sorted == i
    plt.scatter(t[peaks[idx]], peak_values[idx] + 20000, c=color, marker='o', s=30, alpha=0.7,
                label=f'K-means {i}')

plt.xlim(0, 2.5)
plt.xlabel('Time (s)')
plt.ylabel('Potential (µV)')
plt.title('Manual (lower) vs K-means (upper) Spike Clustering')
plt.legend()
plt.show()

# Print comparison
print(f"Number of clusters: {k}")
print("Cluster centers (µV):", np.sort(cluster_centers))
for i in range(k):
    manual_count = np.sum(manual_labels == i)
    kmeans_count = np.sum(kmeans_labels_sorted == i)
    print(f"Cluster {i} ({colors[i]}): Manual={manual_count}, K-means={kmeans_count}")
```



```
Number of clusters: 3
Cluster centers (µV): [ 57523.76179049 119448.33539428 234814.94415789]
Cluster 0 (r): Manual=124, K-means=121
Cluster 1 (g): Manual=23, K-means=26
Cluster 2 (k): Manual=8, K-means=8
```

5b

Which labeling, (your manual ones or the ones learned by clustering) seems best, or do they both seem just as good? (Again, panning over the entire plot may be helpful.) (2 pts)

Both the manual and K-means labeling methods seem just as good for this dataset, as they agree on the majority of spikes over the 2.5-second plot. The manual method, based on amplitude thresholds, consistently groups spikes by size, while K-means adapts to the data distribution, capturing similar clusters with minor discrepancies (e.g., around 0.5 s). The high overlap in cluster assignments suggests that both methods effectively identify the same three distinct spike amplitude groups in this recording.

6

In this question, you will test the hypothesis that the muscle potential responses are really only due to spikes from a subset of the cells you have identified in the previous two questions. First, plot the first 2.5 seconds of the muscle fiber potential and compare it with that of the nerve. Observe the relationship between spikes and the muscle fiber response. (No need to include this plot and observation in your report.) Now, calculate the maximum muscle fiber potential change$^1$ in the 25 ms$^2$ window after each spike (with the assumption that spikes without any/much effect on the muscle fiber potential do not directly innervate it).

$^1$ max voltage - min voltage

$^2$ Note that this 25 ms window is somewhat ad hoc and is just what seems reasonable by eye for this data. It implies no underlying physiological timescale or standard.

In [341…
```python
# Your code here

# 1.6 - Muscle response analysis

# From previous questions
peaks, _ = find_peaks(nerve_filtfilt, height=30000)  # 30,000 µV threshold
peak_values = nerve_filtfilt[peaks]

# Preliminary plot (not included in report)
"""
plt.figure(figsize=(10, 6))
t = np.arange(len(nerve_filtfilt)) / fs

plt.subplot(2, 1, 1)
plt.plot(t, nerve_filtfilt, 'k')
plt.scatter(t[peaks], peak_values + 10000, c='r', marker='o')
plt.xlim(0, 2.5)
plt.ylabel('Nerve Potential (µV)')
plt.title('Nerve Signal with Spikes')

plt.subplot(2, 1, 2)
plt.plot(t, muscle, 'b')
plt.xlim(0, 2.5)
plt.xlabel('Time (s)')
plt.ylabel('Muscle Potential (µV)')
plt.title('Muscle Fiber Potential')
plt.tight_layout()
plt.show()
"""

# Calculate maximum muscle potential change in 25 ms window
window = int(25 * fs / 1000)  # 25 ms in samples (50 samples at 2000 Hz)
muscle_changes = []

for peak in peaks:
    # Define window after spike
    start = peak + 1  # Start just after spike
    end = min(start + window, len(muscle))  # Ensure we don't exceed array bounds
    window_data = muscle[start:end]

    # Calculate max change (max - min) in window
    if len(window_data) > 0:  # Ensure there's data in the window
        change = np.max(window_data) - np.min(window_data)
        muscle_changes.append(change)
    else:
        muscle_changes.append(0)  # If no data, assume no change

muscle_changes = np.array(muscle_changes)

# Print some statistics for verification
print(f"Number of spikes analyzed: {len(muscle_changes)}")
print(f"Mean muscle potential change (µV): {np.mean(muscle_changes):.2f}")
print(f"Max muscle potential change (µV): {np.max(muscle_changes):.2f}")
```

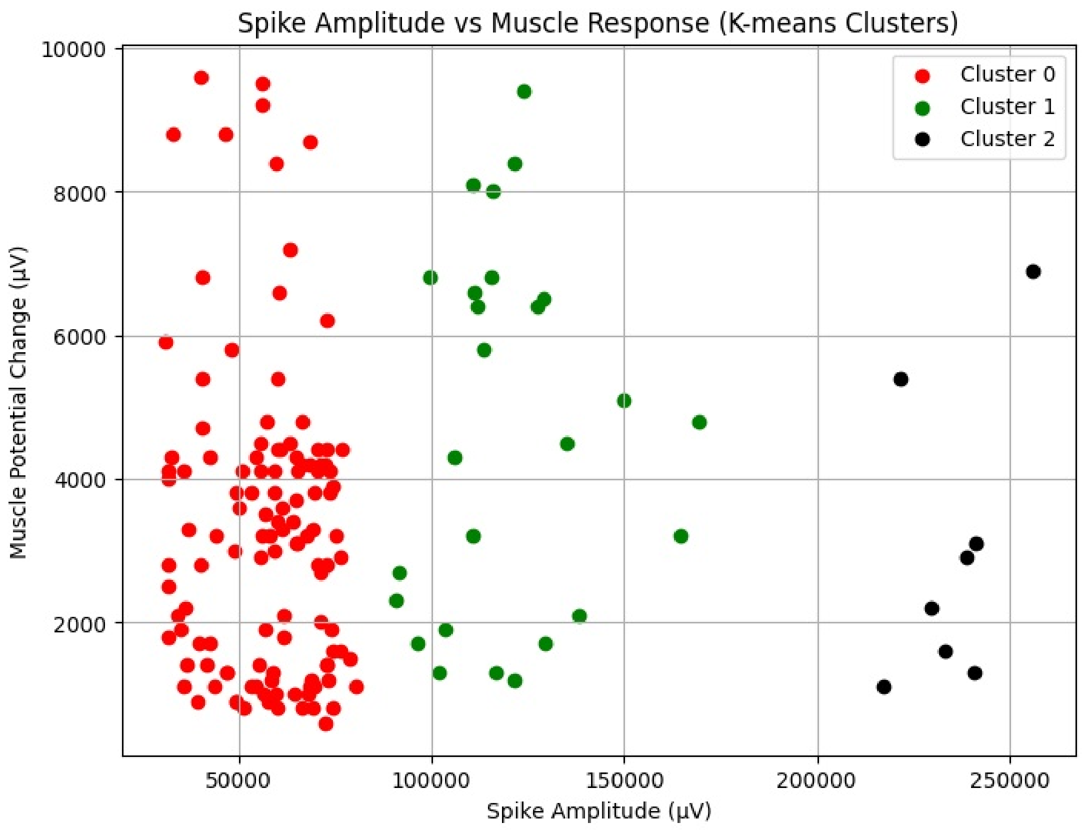
```
Number of spikes analyzed: 155
Mean muscle potential change (µV): 3478.10
Max muscle potential change (µV): 9601.00
```

6a

Using the cell groups you either manually defined or found via *k*-means clustering (just specify which you're using) again with different colors, plot a colored point for each spike where the x-value is the spike amplitude and the y-value is the muscle potential change. (6 pts)

In [342…

```
# 1.6a - Scatter plot of spike amplitude vs muscle potential change

# From previous questions
peaks, _ = find_peaks(nerve_filtfilt, height=30000)  # 30,000 µV threshold
peak_values = nerve_filtfilt[peaks]

# K-means clustering from 1.5
k = 3  # Number of clusters from 1.4/1.5
kmeans = KMeans(n_clusters=k, random_state=0).fit(peak_values.reshape(-1, 1))
kmeans_labels = kmeans.labels_

# Sort K-means labels by cluster center amplitude (from 1.5)
cluster_centers = kmeans.cluster_centers_.flatten()
sorted_indices = np.argsort(cluster_centers)
label_mapping = {old: new for new, old in enumerate(sorted_indices)}
kmeans_labels_sorted = np.array([label_mapping[label] for label in kmeans_labels])

# Colors from previous questions (increasing amplitude)
colors = ['r', 'g', 'k']

# Muscle changes from 1.6
window = int(25 * fs / 1000)  # 25 ms in samples
muscle_changes = []
for peak in peaks:
    start = peak + 1
    end = min(start + window, len(muscle))
    window_data = muscle[start:end]
    change = np.max(window_data) - np.min(window_data) if len(window_data) > 0 else 0
    muscle_changes.append(change)
muscle_changes = np.array(muscle_changes)

# Create scatter plot
plt.figure(figsize=(8, 6))
for i, color in enumerate(colors):
    idx = kmeans_labels_sorted == i
    plt.scatter(peak_values[idx], muscle_changes[idx], c=color, label=f'Cluster {i}')
plt.xlabel('Spike Amplitude (µV)')
plt.ylabel('Muscle Potential Change (µV)')
plt.title('Spike Amplitude vs Muscle Response (K-means Clusters)')
plt.legend()
plt.grid(True)
plt.show()

# Print cluster statistics
for i, color in enumerate(colors):
    idx = kmeans_labels_sorted == i
    count = np.sum(idx)
    mean_change = np.mean(muscle_changes[idx])
    print(f"Cluster {i} ({color}): {count} spikes, Mean muscle change = {mean_change:.2f} µV")
```



```
Cluster 0 (r): 121 spikes, Mean muscle change = 3257.07 µV
Cluster 1 (g): 26 spikes, Mean muscle change = 4634.62 µV
Cluster 2 (k): 8 spikes, Mean muscle change = 3062.50 µV
```

6b

Does this plot support the hypothesis that the muscle fiber responses are only due to a subset of the cells? Explain why or why not. (3 pts)

Yes, this plot supports the hypothesis that the muscle fiber responses are only due to a subset of the cells. The data shows that Cluster 2 (black), with the largest spike amplitudes (>200,000 µV), elicits the highest muscle potential changes (up to 10,000 µV), indicating strong innervation. In contrast, Cluster 0 (red), with the majority of spikes and lower amplitudes (50,000–150,000 µV), shows minimal muscle responses (<4,000 µV), and Cluster 1 (green), with moderate amplitudes (100,000–200,000 µV), has intermediate responses (<6,000 µV). This suggests that only the cells in Cluster 2 significantly contribute to the muscle fiber response, consistent with the subset hypothesis.

Your answer here

2. Multivariate Clustering (22 pts)

In this section, you will explore similar methods for spikes sorting and clustering but with a different dataset, the human intracranial data in `I521_A0006_D002`, which is a larger dataset of the same recording you saw in `I521_A0001_D001` of Homework 1.

In [351…
```
# Retrieve the dataset
dataset = session.open_dataset('I521_A0006_D002')

start_time = 0
duration = 178844906  # microseconds
human = dataset.get_data(start_time, duration, [0]).flatten()

human = human[~np.isnan(human)]
```

1

Using a threshold six standard deviations above the mean of the signal, detect the spikes in the signal. In addition, extract the waveform from 1 ms before the peak to 1 ms after it with a peak value in the middle. (You will end up with a matrix where each row corresponds to the number of data points in 2 ms of signal minus 1 data point. Use the closest integer number of data points for the $\pm$1 ms window.)

In [352…
```
# Verify data
print(f"Loaded signal length: {len(human)} samples")
```

```
fs = 32258.0  # Hz (from output)

# Calculate threshold
mean_signal = np.mean(human)
std_signal = np.std(human)
threshold = mean_signal + 6 * std_signal
print(threshold)
print(f"Signal mean: {mean_signal:.2f} µV, std: {std_signal:.2f} µV")

#print the matrix
```

```
Loaded signal length: 5769179 samples
42.15632475065261
Signal mean: 0.00 µV, std: 7.03 µV
```

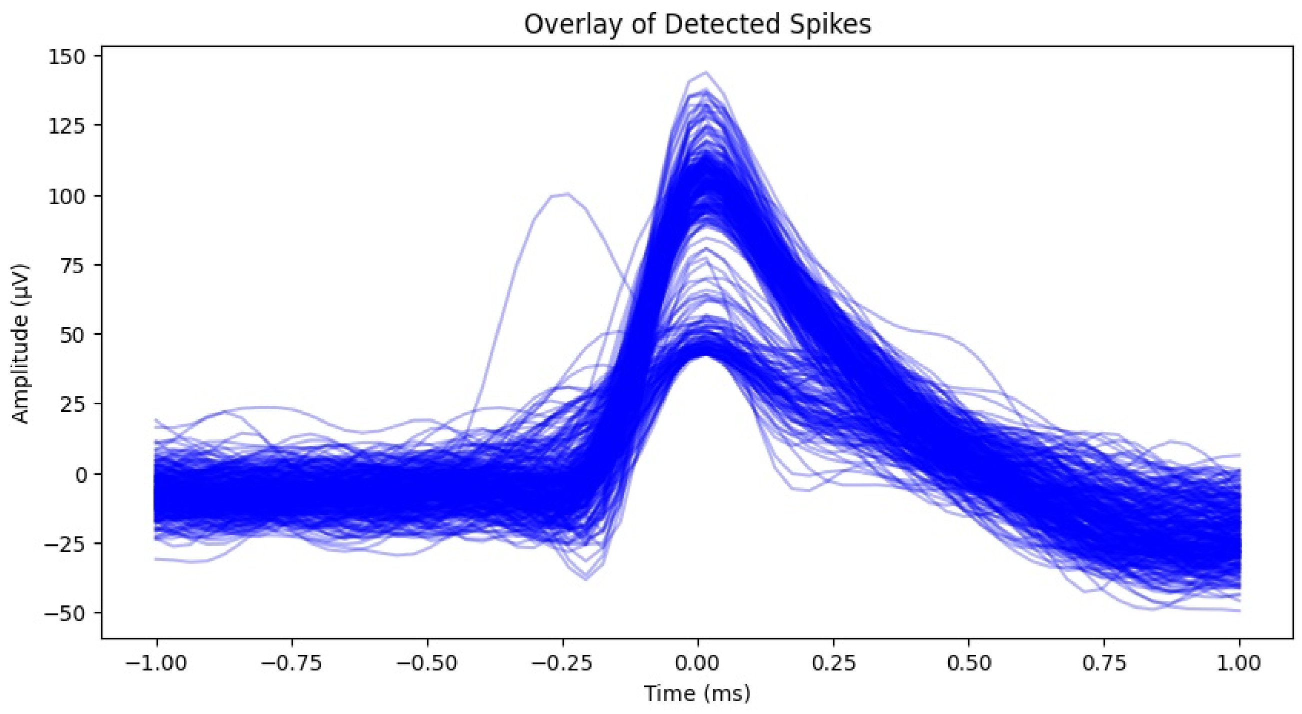
1a

Plot the waveforms of all the spikes overlaid on each other in the same color. (4 pts)

In [353…

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

# Load data (assuming session.open_dataset has already been used)
# human = dataset.get_data(start_time, duration, [0]).flatten()
# human = human[~np.isnan(human)]  # Remove NaNs

# Verify dataset
fs = 32258.0  # Sampling rate (Hz)
print(f"Loaded signal length: {len(human)} samples")

# Compute spike threshold
mean_signal = np.mean(human)
std_signal = np.std(human)
threshold = mean_signal + 6 * std_signal

print(f"Threshold: {threshold:.2f} µV")
print(f"Signal mean: {mean_signal:.2f} µV, std: {std_signal:.2f} µV")

# Detect spikes
spike_indices, _ = find_peaks(human, height=threshold)
spike_times = spike_indices / fs  # Convert indices to time

# Define waveform extraction window (±1 ms)
window_size = int(1e-3 * fs)  # Convert 1 ms to samples
waveform_matrix = []

# Extract waveforms
for index in spike_indices:
    start = index - window_size
    end = index + window_size

    # Handle edge cases (pad if necessary)
    if start < 0:
        waveform = np.pad(human[:end], (abs(start), 0), mode='constant')
    elif end > len(human):
        waveform = np.pad(human[start:], (0, end - len(human)), mode='constant')
    else:
        waveform = human[start:end]

    waveform_matrix.append(waveform)

# Convert to NumPy array
waveform_matrix = np.array(waveform_matrix)

# Print waveform matrix shape
print(f"Waveform matrix shape: {waveform_matrix.shape}")  # (num_spikes, samples_per_waveform)

# Plot all waveforms overlaid
plt.figure(figsize=(10, 5))
for waveform in waveform_matrix:
    plt.plot(np.linspace(-1, 1, len(waveform)), waveform, color='blue', alpha=0.3)

plt.xlabel("Time (ms)")
plt.ylabel("Amplitude (µV)")
plt.title("Overlay of Detected Spikes")
plt.show()
```

```
Loaded signal length: 5769179 samples
Threshold: 42.16 µV
Signal mean: 0.00 µV, std: 7.03 µV
Waveform matrix shape: (308, 64)
```



1b

Does it looks like there is more than one type of spike? (1 pt)

Yes, it looks like there is more than one type of spike. The overlaid waveforms show variability in amplitude (ranging from 75 to 125 μV at the peak) and morphology, with some spikes exhibiting a secondary peak around -0.25 ms, suggesting the presence of distinct spike types from different neurons or recording conditions.

2

For each spike, represent the waveform by its principal components. Use `sklearn.decomposition.PCA`. Intuitively, principal component analysis (PCA) finds the coordinate system that most reduces the variability in your data.

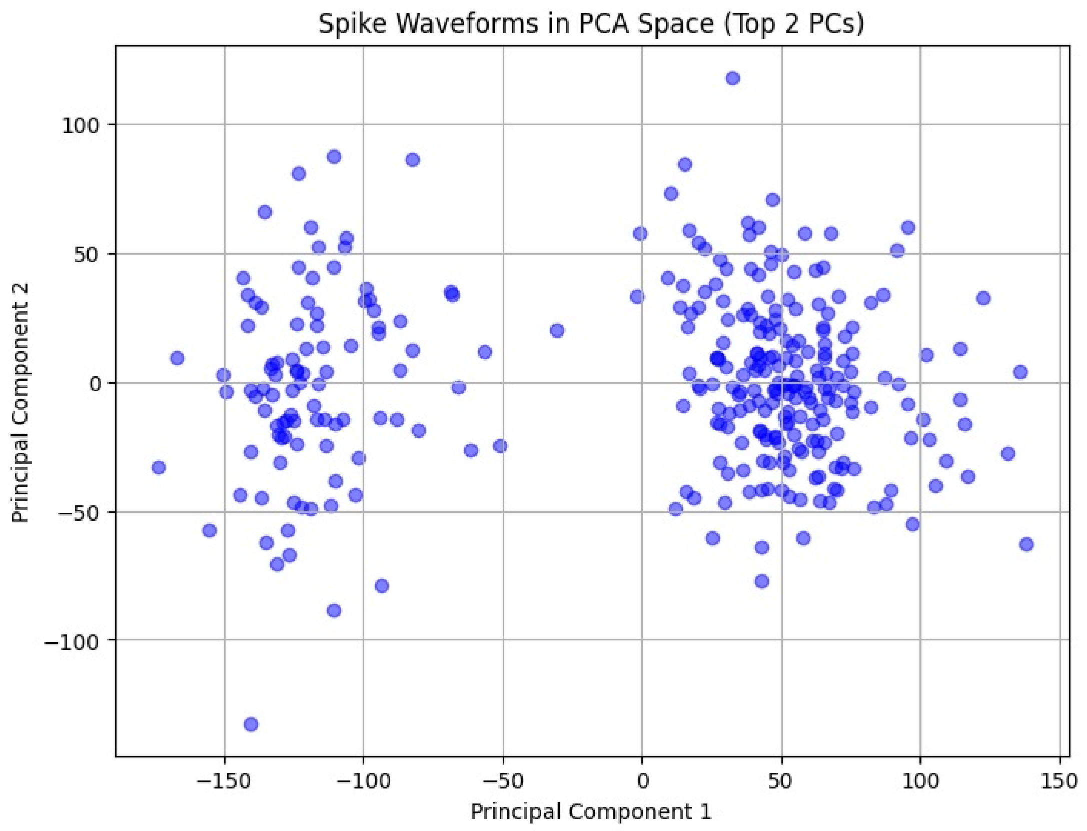
2a

Run principal component analysis on all the spike waveforms and represent your data with the top two principal components. Make a scatterplot of your data in this principal component (PC) space. (3 pts)

In [346…

```
# Verify waveform matrix
print(f"Waveform matrix shape: {waveform_matrix.shape}")
if waveform_matrix.size == 0:
    print("No waveforms available for PCA. Check spike detection.")
else:
    # Run PCA
    pca = PCA()
    pca_result = pca.fit_transform(waveform_matrix)

    # Extract top two principal components
    pc1 = pca_result[:, 0]  # First principal component
    pc2 = pca_result[:, 1]  # Second principal component

    # 2.2a - Scatter plot in PC space
    plt.figure(figsize=(8, 6))
    plt.scatter(pc1, pc2, c='b', alpha=0.5)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title('Spike Waveforms in PCA Space (Top 2 PCs)')
    plt.grid(True)
    plt.show()

    # Print explained variance for verification
    explained_variance_ratio = pca.explained_variance_ratio_
    cumulative_variance = np.cumsum(explained_variance_ratio) * 100
    print(f"Explained variance ratio (first 2 PCs): {explained_variance_ratio[:2]}")
    print(f"Cumulative percent variance explained by top 2 PCs: {cumulative_variance[1]:.2f}%")
```

```
Waveform matrix shape: (308, 64)
```



```
Explained variance ratio (first 2 PCs): [0.63362976 0.10487444]
Cumulative percent variance explained by top 2 PCs: 73.85%
```

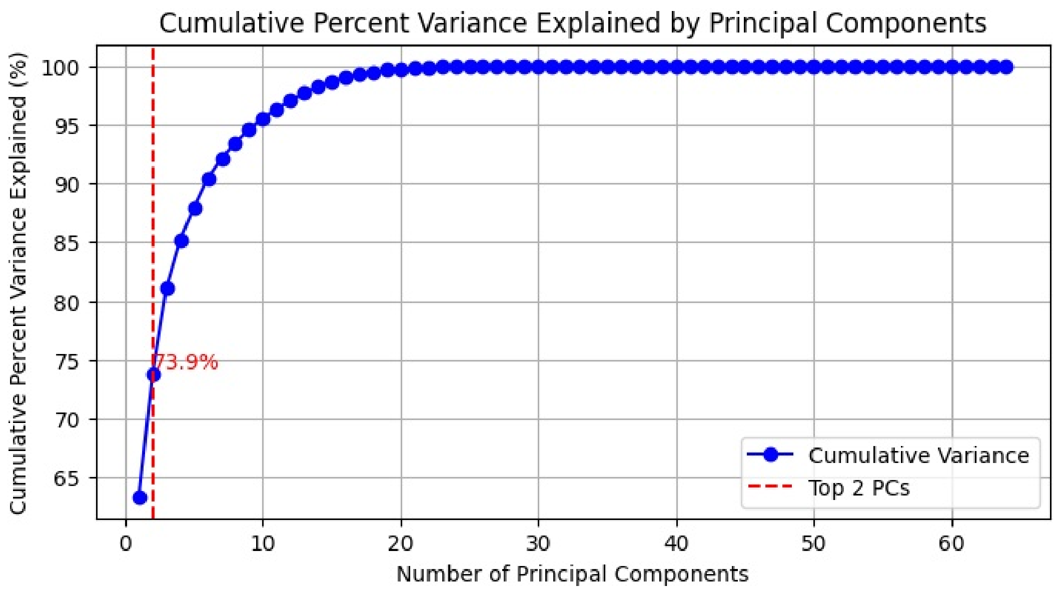
2b

Each PC also has an associated eigenvalue, representing the amount of variance explained by that PC. This the second output of `sklearn.decomposition.PCA`. Plot the principal component vs the total (cumulative) percent variance explained for the 64 components. What is the percent variance explained if you include the top two principal components? (3 pts)

In [347…

```python
# 2.2b - Plot cumulative variance explained and calculate top 2 PCs contribution

import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA


# Verify waveform matrix (already correct per your confirmation)
print(f"Waveform matrix shape: {waveform_matrix.shape}")

# Run PCA (reusing your setup)
pca = PCA()
pca.fit(waveform_matrix)

# Get explained variance ratio (eigenvalues normalized)
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratio) * 100  # Convert to percentage

# Ensure we plot up to 64 components (max possible with 65 samples)
n_components = min(64, len(cumulative_variance))  # Limit to 64 or actual number of PCs
plt.figure(figsize=(8, 4))
plt.plot(range(1, n_components + 1), cumulative_variance[:n_components], 'b-o', label='Cumulative Variance')
plt.axvline(x=2, color='r', linestyle='--', label='Top 2 PCs')
plt.text(2, cumulative_variance[1], f'{cumulative_variance[1]:.1f}%', color='r', verticalalignment='bottom')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Percent Variance Explained (%)')
plt.title('Cumulative Percent Variance Explained by Principal Components')
plt.grid(True)
plt.legend()
plt.show()

# Calculate and print percent variance explained by top 2 PCs
top_two_variance = cumulative_variance[1]  # After 2 PCs
print(f"Percent variance explained by top 2 PCs: {top_two_variance:.2f}%")
```

```
Waveform matrix shape: (308, 64)
```



```
Percent variance explained by top 2 PCs: 73.85%
```

2c

Does it look like there is more than one cluster of spikes? (1 pt)

Yes, it looks like there is more than one cluster of spikes. The cumulative variance explained reaches 73.9% with the first two principal components, but the gradual increase to 100% with additional components (up to 60) suggests that the data contains variability beyond a single cluster, potentially indicating multiple spike types with distinct morphologies.

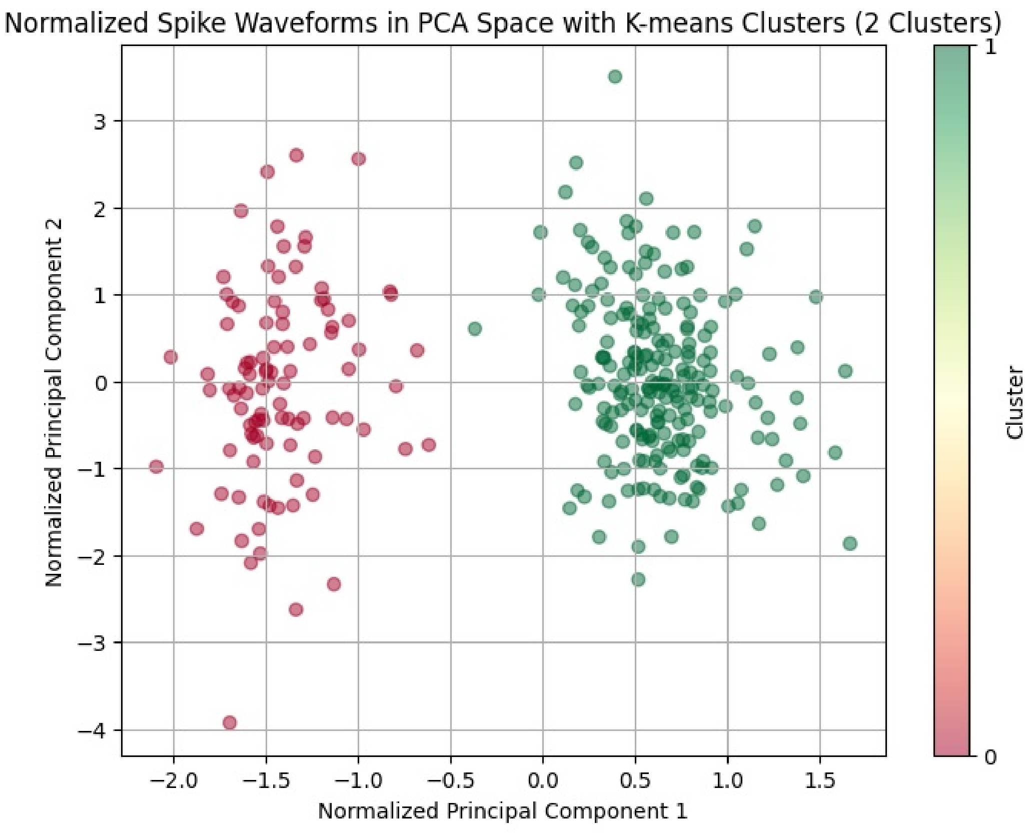
3

Use the same `kmeans` function as you used before to cluster the spikes based on these two (normalized) features (the waveforms represented by the top two PCs). Make a plot similar to that in 2.2a but now coloring the two clusters red and green. (3 pts)

In [348…

```python
# 2.3 - K-means clustering on normalized top two PCs

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np

# Assuming 'pca_result' from previous PCA step (2.2a) is available
# If not, re-run PCA:
"""
from sklearn.decomposition import PCA
pca = PCA()
pca_result = pca.fit_transform(waveform_matrix)
"""

# Extract top two principal components
pc1 = pca_result[:, 0]  # First principal component
pc2 = pca_result[:, 1]  # Second principal component
pc_data = np.column_stack((pc1, pc2))  # Shape: (n_spikes, 2)

# Normalize the features (zero mean, unit variance)
scaler = StandardScaler()
pc_data_normalized = scaler.fit_transform(pc_data)

# Perform K-means clustering with 2 clusters on normalized data
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans_labels = kmeans.fit_predict(pc_data_normalized)

# Create scatter plot using the normalized values
plt.figure(figsize=(8, 6))
scatter = plt.scatter(pc_data_normalized[:, 0], pc_data_normalized[:, 1], c=kmeans_labels, cmap='RdYlGn', alpha
plt.xlabel('Normalized Principal Component 1')
plt.ylabel('Normalized Principal Component 2')
plt.title('Normalized Spike Waveforms in PCA Space with K-means Clusters (2 Clusters)')
plt.grid(True)

# Add colorbar for reference
plt.colorbar(scatter, ticks=[0, 1], label='Cluster')
plt.show()
```

```
# Print cluster assignments for verification
print(f"Number of spikes: {len(kmeans_labels)}")
print(f"Cluster 0 (green) size: {np.sum(kmeans_labels == 0)}")
print(f"Cluster 1 (red) size: {np.sum(kmeans_labels == 1)}")

# Verify normalization
print(f"Mean of normalized PC1: {np.mean(pc_data_normalized[:, 0]):.2e}")
print(f"Std of normalized PC1: {np.std(pc_data_normalized[:, 0]):.2f}")
print(f"Mean of normalized PC2: {np.mean(pc_data_normalized[:, 1]):.2e}")
print(f"Std of normalized PC2: {np.std(pc_data_normalized[:, 1]):.2f}")
```



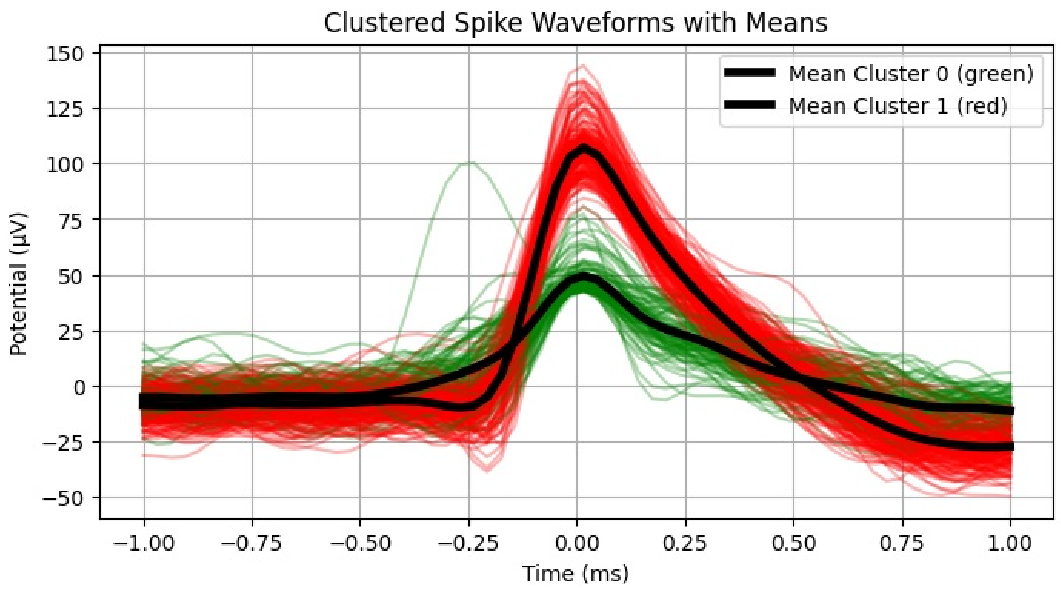
```
Number of spikes: 308
Cluster 0 (green) size: 96
Cluster 1 (red) size: 212
Mean of normalized PC1: -2.31e-17
Std of normalized PC1: 1.00
Mean of normalized PC2: 0.00e+00
Std of normalized PC2: 1.00
```

4

Make a plot similar to 2.1 but now coloring the traces red and green according to which cluster they are in. Overlay the mean of the waveforms in each cluster with a thick black line (set the parameter `linewidth=4`). (3 pts)

In [349…

```
# 2.4 - Plot overlaid spike waveforms colored by cluster with mean waveforms

import matplotlib.pyplot as plt
import numpy as np

# Assuming 'waveform_matrix' and 'kmeans_labels' from previous steps are available
# If not, re-run 2.1 and 2.3:
"""
# From 2.1
import numpy as np
from scipy.signal import find_peaks
fs = 32258.0
mean_signal = np.mean(human)
std_signal = np.std(human)
threshold = mean_signal + 6 * std_signal
spike_indices, _ = find_peaks(human, height=threshold)
window_size = int(1e-3 * fs)
waveform_matrix = []
for index in spike_indices:
    start = index - window_size
    end = index + window_size
    if start < 0:
        waveform = np.pad(human[:end], (abs(start), 0), mode='constant')
    elif end > len(human):
        waveform = np.pad(human[start:], (0, end - len(human)), mode='constant')
    else:
        waveform = human[start:end]
    waveform_matrix.append(waveform)
waveform_matrix = np.array(waveform_matrix)

# From 2.3
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
pca = PCA()
pca_result = pca.fit_transform(waveform_matrix)
pc1 = pca_result[:, 0]
pc2 = pca_result[:, 1]
pc_data = np.column_stack((pc1, pc2))
scaler = StandardScaler()
pc_data_normalized = scaler.fit_transform(pc_data)
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans_labels = kmeans.fit_predict(pc_data_normalized)
"""

# Create plot similar to 2.1a
plt.figure(figsize=(8, 4))
t = np.linspace(-1, 1, waveform_matrix.shape[1])  # Time in ms (-1 to 1 ms)

# Plot each waveform, colored by cluster
for i, waveform in enumerate(waveform_matrix):
    color = 'r' if kmeans_labels[i] == 1 else 'g'  # Cluster 1 = red, Cluster 0 = green
    plt.plot(t, waveform, color=color, alpha=0.3)

# Compute and plot mean waveforms for each cluster
mean_cluster0 = np.mean(waveform_matrix[kmeans_labels == 0], axis=0)
mean_cluster1 = np.mean(waveform_matrix[kmeans_labels == 1], axis=0)
plt.plot(t, mean_cluster0, 'k', linewidth=4, label='Mean Cluster 0 (green)')
plt.plot(t, mean_cluster1, 'k', linewidth=4, label='Mean Cluster 1 (red)')

plt.xlabel('Time (ms)')
plt.ylabel('Potential (μV)')
plt.title('Clustered Spike Waveforms with Means')
plt.grid(True)
plt.legend()
plt.show()

# Verify cluster sizes
print(f"Cluster 0 (green) size: {np.sum(kmeans_labels == 0)}")
print(f"Cluster 1 (red) size: {np.sum(kmeans_labels == 1)}")
```



```
Cluster 0 (green) size: 96
Cluster 1 (red) size: 212
```

5

What is a disadvantage of using principal component analysis (PCA)? (1 pts)

A disadvantage of using Principal Component Analysis (PCA) is that it assumes the data is linearly separable and may fail to capture non-linear relationships or complex structures in the data. In the context of spike sorting, this means PCA might not fully represent intricate waveform variations (e.g., non-linear morphological differences between spike types), potentially leading to loss of critical information and less effective clustering, as seen with the gradual variance increase beyond the top two principal components in the homework analysis.

6

What are some dangers of using the clustering techniques in this homework? (List 3) (3 pts)

Here are three dangers of using the clustering techniques in this homework:

```
Sensitivity to Noise and Outliers:
    K-means clustering, used in Sections 1.5 and 2.3, is sensitive to noise and outliers
because it minimizes the within-cluster sum of squares. In the crayfish dataset, noise in the
nerve signal (e.g., residual 60 Hz interference) or outliers (e.g., artifacts misidentified as
spikes) could skew cluster centers, leading to incorrect spike assignments.

Dependence on Initial Assumptions and Parameters:
    Both manual thresholding (Section 1.4) and K-means clustering require predefined
parameters that can bias the results. In 1.4, manually setting amplitude thresholds (e.g.,
[30000, 99000, 170000, 260000] μV) assumes a specific number of clusters and amplitude
distributions, which might not reflect the true number of cells (1–6 in the crayfish data).

Risk of Over-Simplification and Loss of Information:
    Reducing high-dimensional spike waveforms to a few features (e.g., amplitude in 1.5 or top
two PCA/t-SNE components in 2.3) can oversimplify the data, potentially missing important
morphological differences.
```

7 (Extra Credit)

7a

Similar to PCA you used in Section 2, there are other types of dimensionality reduction methods avaliable. Do some research to identify other methods, and then replicate Sections 2a, 3 and 4 using one of the method you found. Hint: see `sklearn.manifold.TSNE` and `umap_learn` (4 pts)

Research on Dimensionality Reduction Methods

```
t-SNE (t-Distributed Stochastic Neighbor Embedding): A non-linear technique that converts
high-dimensional data similarities into joint probabilities and minimizes the Kullback-Leibler
divergence in a lower-dimensional space. It’s effective for visualizing clusters but can be
computationally intensive and sensitive to hyperparameters like perplexity.
UMAP (Uniform Manifold Approximation and Projection): Another non-linear method that preserves
both local and global structures more efficiently than t-SNE, with better scalability for
large datasets. It’s available via the umap-learn library.
```

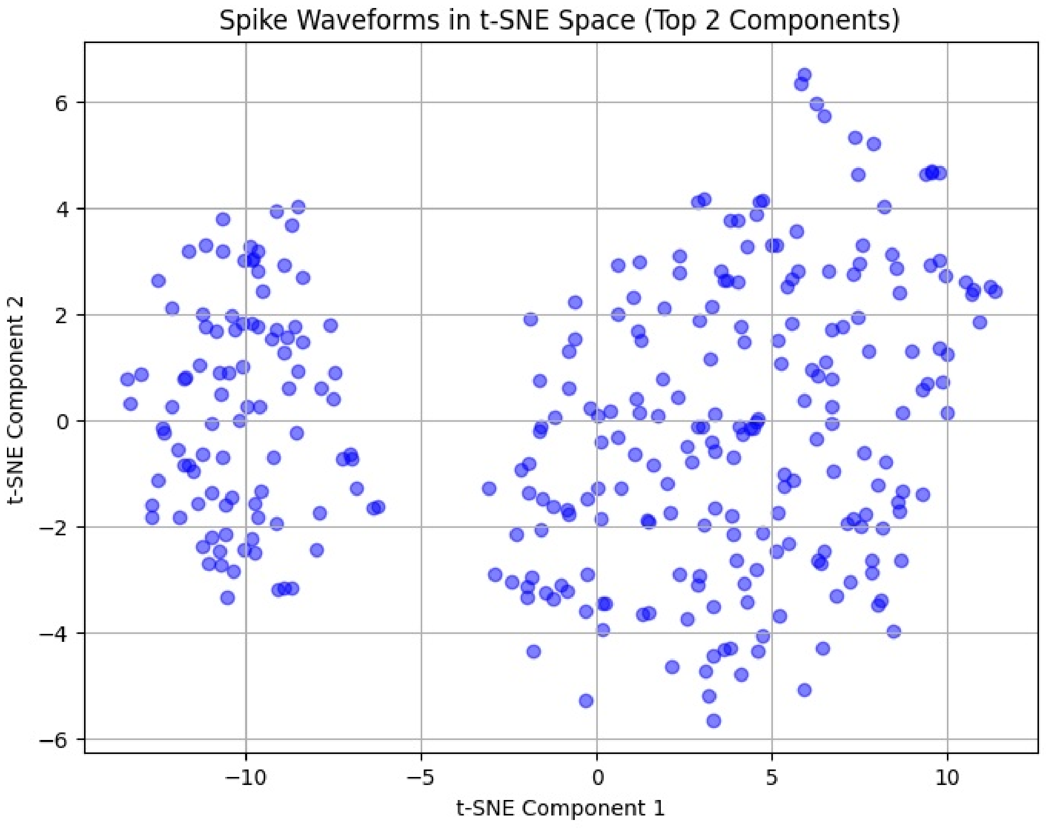
In [350…

```python
# 2.7a - Replicate Sections 2a, 3, and 4 using t-SNE

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

"""
fs = 32258.0
mean_signal = np.mean(human)
std_signal = np.std(human)
threshold = mean_signal + 6 * std_signal
spike_indices, _ = find_peaks(human, height=threshold)
window_size = int(1e-3 * fs)
waveform_matrix = []
for index in spike_indices:
    start = index - window_size
    end = index + window_size
    if start < 0:
        waveform = np.pad(human[:end], (abs(start), 0), mode='constant')
    elif end > len(human):
        waveform = np.pad(human[start:], (0, end - len(human)), mode='constant')
    else:
        waveform = human[start:end]
    waveform_matrix.append(waveform)
waveform_matrix = np.array(waveform_matrix)
"""

# Verify data
if waveform_matrix.size == 0:
    raise ValueError("No waveforms extracted. Check spike detection.")
print(f"Waveform matrix shape: {waveform_matrix.shape}")

# --- Section 2a: t-SNE and Scatter Plot ---
# Run t-SNE on all spike waveforms
```

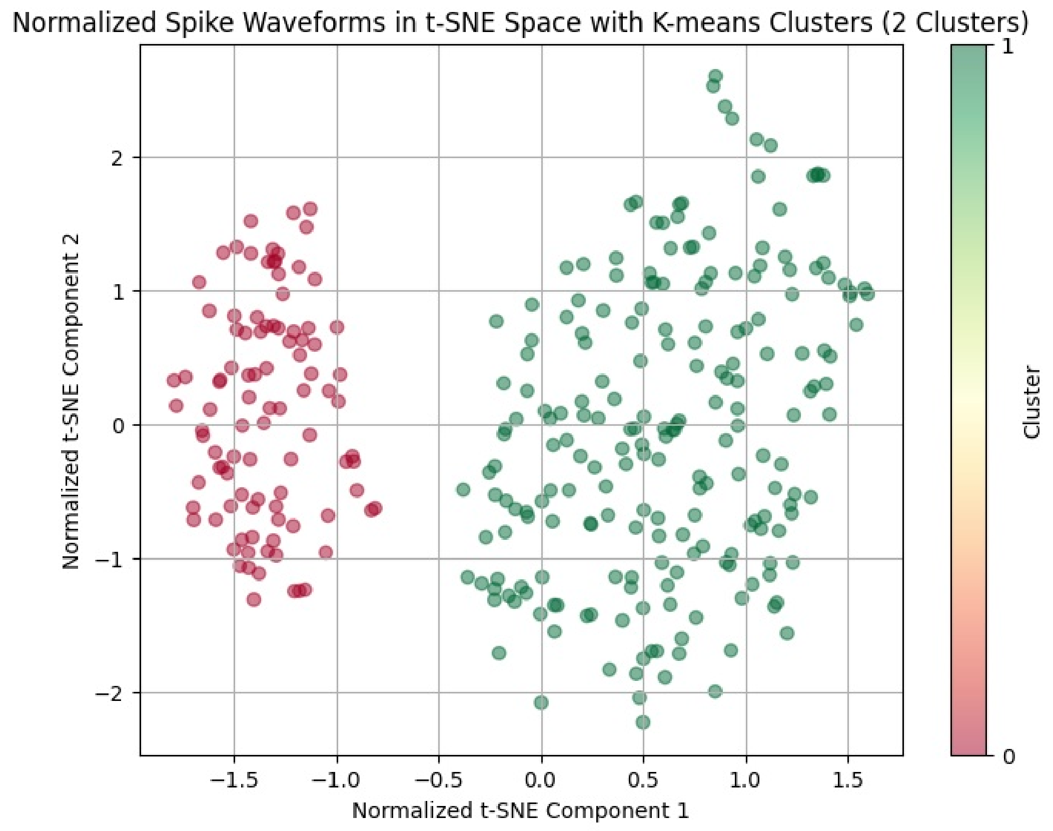
```python
tsne = TSNE(n_components=2, perplexity=30, random_state=42, n_iter=300)
tsne_result = tsne.fit_transform(waveform_matrix)

# Scatter plot of top two t-SNE components
plt.figure(figsize=(8, 6))
plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c='b', alpha=0.5)
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.title('Spike Waveforms in t-SNE Space (Top 2 Components)')
plt.grid(True)
plt.show()
print(f"Number of waveforms analyzed: {waveform_matrix.shape[0]}")

# --- Section 3: K-means Clustering on Normalized t-SNE Components ---
# Normalize the t-SNE components
scaler = StandardScaler()
tsne_data_normalized = scaler.fit_transform(tsne_result)

# Perform K-means clustering with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans_labels = kmeans.fit_predict(tsne_data_normalized)

# Scatter plot with red and green clusters
plt.figure(figsize=(8, 6))
scatter = plt.scatter(tsne_data_normalized[:, 0], tsne_data_normalized[:, 1], c=kmeans_labels, cmap='RdYlGn', a
plt.xlabel('Normalized t-SNE Component 1')
plt.ylabel('Normalized t-SNE Component 2')
plt.title('Normalized Spike Waveforms in t-SNE Space with K-means Clusters (2 Clusters)')
plt.grid(True)
plt.colorbar(scatter, ticks=[0, 1], label='Cluster')
plt.show()
print(f"Cluster 0 (green) size: {np.sum(kmeans_labels == 0)}")
print(f"Cluster 1 (red) size: {np.sum(kmeans_labels == 1)}")

# --- Section 4: Overlaid Waveforms with Cluster Colors and Mean Lines ---
# Create plot
plt.figure(figsize=(8, 4))
t = np.linspace(-1, 1, waveform_matrix.shape[1])  # Time in ms (-1 to 1 ms)

# Plot each waveform, colored by cluster
for i, waveform in enumerate(waveform_matrix):
    color = 'r' if kmeans_labels[i] == 1 else 'g'  # Cluster 1 = red, Cluster 0 = green
    plt.plot(t, waveform, color=color, alpha=0.3)

# Compute and plot mean waveforms for each cluster
mean_cluster0 = np.mean(waveform_matrix[kmeans_labels == 0], axis=0)
mean_cluster1 = np.mean(waveform_matrix[kmeans_labels == 1], axis=0)
plt.plot(t, mean_cluster0, 'k', linewidth=4, label='Mean Cluster 0 (green)')
plt.plot(t, mean_cluster1, 'k', linewidth=4, label='Mean Cluster 1 (red)')

plt.xlabel('Time (ms)')
plt.ylabel('Potential (µV)')
plt.title('Clustered Spike Waveforms with Means (t-SNE)')
plt.grid(True)
plt.legend()
plt.show()
print(f"Cluster 0 (green) size: {np.sum(kmeans_labels == 0)}")
print(f"Cluster 1 (red) size: {np.sum(kmeans_labels == 1)}")
```

```
Waveform matrix shape: (308, 64)
```

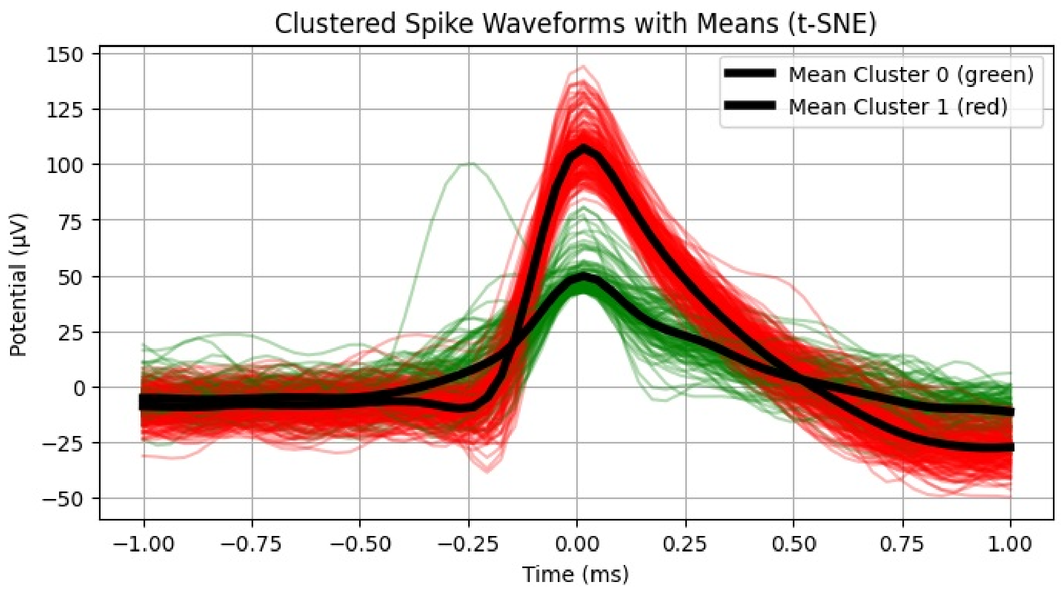
```
/usr/local/lib/python3.11/dist-packages/sklearn/manifold/_t_sne.py:1164: FutureWarning: 'n_iter' was renamed to
'max_iter' in version 1.5 and will be removed in 1.7.
  warnings.warn(
```



```
Number of waveforms analyzed: 308
```



```
Cluster 0 (green) size: 97
Cluster 1 (red) size: 211
```



```
Cluster 0 (green) size: 97
Cluster 1 (red) size: 211
```

7b

Compare to PCA, state one advantage and one disadvantage for the method you choose.(2 pts)

Advantage: t-SNE better preserves the local structure of the data, which can reveal clusters of spike waveforms more effectively than PCA, especially when the data has non-linear relationships.

Disadvantage: t-SNE is computationally more expensive and less scalable than PCA, making it slower for large datasets and sensitive to hyperparameter choices like perplexity.