```python
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy import io
from scipy.signal import butter, lfilter, freqz, medfilt

def load_mat_files(dataDir):
    mats = []
    for file in os.listdir(dataDir):
        mats.append(io.loadmat(dataDir+file)['gestures'])
    return mats

def butter_bandpass_filter(data, lowcut=20.0, highcut=400.0, fs=2048, order=4):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    y = lfilter(b, a, data)
    return y

def plot_bandpass_filtered_data(data):
    plt.figure(1)
    plt.clf()
    plt.plot(data, label='Noisy signal')

    y = butter_bandpass_filter(data)
    plt.plot(y, label='Filtered signal')
    plt.xlabel('time (seconds)')
    plt.grid(True)
    plt.axis('tight')
    plt.legend(loc='upper left')
    plt.show()

def divide_to_windows(datas, window_size=150):
    windows=np.delete(datas,
list(range((len(datas)//window_size)*window_size,len(datas))))
    windows=np.reshape(windows,((len(datas)//window_size,window_size)))
    return windows

def compute_RMS(datas):
    return np.sqrt(np.mean(datas**2))

def compute_RMS_for_each_windows(windows):
    init=1
    for window in windows:
        if init==1:
            RMSs=np.array([[compute_RMS(window)]])
            init=0
            continue
        RMSs=np.append(RMSs, [[compute_RMS(window)]], axis=0)
    return RMSs

def create_168_dimensional_window_vectors(channels):
    for i_ch in range(len(channels)):
        # Segmentation : Data processing : Discard useless data
        if (i_ch+1)%8 == 0:
            continue
        # Preprocessing : Apply butterworth band-pass filter]
        filtered_channel=butter_bandpass_filter(channels[i_ch])
```

```
        # Segmentation : Data processing : Divide continuous data into 150 samples window
        windows_per_channel=divide_to_windows(filtered_channel)
        # Segmentation : Compute RMS for each channel
        RMSwindows_per_channel=compute_RMS_for_each_windows(windows_per_channel)
        if i_ch==0:
            RMS_one_try=np.array(RMSwindows_per_channel)
            continue
        RMS_one_try=np.append(RMS_one_try, RMSwindows_per_channel, axis=1)  # Adding column
    return RMS_one_try

def average_for_channel(gesture):
    average=np.array([])
    for i_ch in range(gesture.shape[2]):
        sum=0
        for i_win in range(gesture.shape[1]):
            for i_try in range(gesture.shape[0]):
                sum+=gesture[i_try][i_win][i_ch]
        average=np.append(average, [sum/(gesture.shape[1]*gesture.shape[0])])
    return average

def base_normalization(RMS_gestures):
    average_channel_idle_gesture=average_for_channel(RMS_gestures[0])
    for i_ges in range(RMS_gestures.shape[0]):   # Including idle gesture
        for i_try in range(RMS_gestures.shape[1]):
            for i_win in range(RMS_gestures.shape[2]):
                for i_ch in range(RMS_gestures.shape[3]):
                    RMS_gestures[i_ges][i_try][i_win][i_ch]-=average_channel_idle_gesture[i_ch]
    return RMS_gestures

def ACTIVE_filter(RMS_gestures):
    for i_ges in range(len(RMS_gestures)):
        for i_try in range(len(RMS_gestures[i_ges])):
            # Segmentation : Determine whether ACTIVE : Compute summarized RMS
            sum_RMSs=[sum(window) for window in RMS_gestures[i_ges][i_try]]
            threshold=sum(sum_RMSs)/len(sum_RMSs)
            # Segmentation : Determine whether ACTIVE
            i_ACTIVEs=[]
            for i_win in range(len(RMS_gestures[i_ges][i_try])):
                if sum_RMSs[i_win] > threshold:
                    i_ACTIVEs.append(i_win)
            for i in range(len(i_ACTIVEs)):
                if i==0:
                    continue
                if i_ACTIVEs[i]-i_ACTIVEs[i-1] == 2:
                    i_ACTIVEs.insert(i, i_ACTIVEs[i-1]+1)
            # Segmentation : Determine whether ACTIVE : Select the longest contiguous sequences
            segs=[]
            contiguous = 0
            for i in range(len(i_ACTIVEs)):
                if i == len(i_ACTIVEs)-1:
                    if contiguous!=0:
                        segs.append((start, contiguous))
                    break
                if i_ACTIVEs[i+1]-i_ACTIVEs[i] == 1:
                    if contiguous == 0:
                        start=i_ACTIVEs[i]
```

```
                    contiguous+=1
                else:
                    if contiguous != 0:
                        contiguous+=1
                        segs.append((start, contiguous))
                        contiguous=0
            seg_start, seg_len = sorted(segs, key=lambda seg: seg[1], reverse=True)[0]
            # Segmentation : Determine whether ACTIVE : delete if the window is not ACTIVE
            for i_win in reversed(range(len(RMS_gestures[i_ges][i_try]))):
                if not i_win in range(seg_start, seg_start+seg_len):
                    del RMS_gestures[i_ges][i_try][i_win]
    return RMS_gestures

def check(x):
    print("length: ", len(x))
    print("type: ", type(x))
    raise ValueError("------------WORKING LINE-------------")

def main():
    #loading .mat files consist of 0,1,2,3(,11,17,18,21,23,24,25 not for light) gestures
    gestures = load_mat_files("../data/ref1_subject1_session1_light/")  # gestures : list
    #In idle gesture, we just use 2,4,7,8,11,13,19,25,26,30th tries in order to match the number of datas
    gestures[0]=gestures[0][[1,3,6,7,10,12,18,24,25,29]]

    # Signal Preprocessing & Data processing for segmentation
    init_gesture=1
    for gesture in gestures:
        init_try=1
        for one_try in gesture:
            RMS_one_try = create_168_dimensional_window_vectors(one_try[0]) # one_try[0] : channels, ndarray
            if init_try == 1:
                RMS_tries_for_gesture = np.array([RMS_one_try])
                init_try=0
                continue
            RMS_tries_for_gesture = np.append(RMS_tries_for_gesture, [RMS_one_try], axis=0) # Adding height
        if init_gesture==1:
            RMS_gestures = np.array([RMS_tries_for_gesture])
            init_gesture=0
            continue
        RMS_gestures = np.append(RMS_gestures, [RMS_tries_for_gesture], axis=0) # Adding blocks
    # Segmentation : Data processing : Base normalization
    RMS_gestures=base_normalization(RMS_gestures)
    # Segmentation : Data processing : Median filtering
    ############################### ZERO-PADDING ###################################
    RMS_gestures=medfilt(RMS_gestures, kernel_size=3)
    # Segmentation : Dertermine which window is ACTIVE
    ACTIVE_segments=ACTIVE_filter(RMS_gestures.tolist())

    print("# of gestures: %d" %len(ACTIVE_RMS_gestures))
    print("# of tries: %d" %len(ACTIVE_RMS_gestures[0]))
    print("# of windows: %d" %len(ACTIVE_RMS_gestures[0][0]))
    print("# of channels: %d" %len(ACTIVE_RMS_gestures[0][0][0]))
```

```

main()
```