Help me understand then these functions that utilize the raw eda signals. What do these functions do?

```
def butter_lowpass(cutoff, fs, order=5):
   # Filtering Helper functions
   nyq = 0.5 * fs
   normal_cutoff = cutoff / nyq
   b, a = scisig.butter(order, normal_cutoff, btype='low', analog=False)
   return b, a


def butter_lowpass_filter(data, cutoff, fs, order=5):
   # Filtering Helper functions
   b, a = butter_lowpass(cutoff, fs, order=order)
   y = scisig.lfilter(b, a, data)
   return y
```

expand_more

volume_up

These functions are designed to perform low-pass filtering on a signal using the Butterworth filter design. Here's a breakdown of their functionality:

1. butter_lowpass(cutoff, fs, order=5):

This function defines the Butterworth low-pass filter and returns the filter coefficients.

Arguments:

cutoff: This is the desired cut-off frequency of the filter. It represents the highest frequency that will pass through the filter without significant attenuation. (Units: Hz)

fs: This is the sampling frequency of the EDA signal. It represents the number of samples collected per second. (Units: Hz) order (optional): This is the order of the filter. Higher order filters provide sharper transitions between the passband and stopband but can also introduce more processing delay. (Default: 5)

Functionality:

Nyquist Frequency: It calculates the Nyquist frequency, which is half the sampling frequency. This is the highest frequency that can be represented without aliasing in a discretely sampled signal.

For a given sampling rate (samples per second), the Nyquist frequency (cycles per second) is the frequency whose cycle-length (or period) is twice the interval

In signal processing, the Nyquist rate, named after Harry Nyquist, is a value equal to twice the highest frequency (bandwidth) of a given function or signal

Normalized Cutoff: It normalizes the desired cut-off frequency by dividing it by the Nyquist frequency. This is because the Butterworth filter design operates on normalized frequencies between 0 and 1.

Filter Design: It uses the scipy.signal.butter function to design the Butterworth low-pass filter. This function takes the normalized cut-off frequency, filter order, and filter type ('lowpass' in this case) as arguments. It returns two coefficients:

b: Numerator coefficients of the filter transfer function.

a: Denominator coefficients of the filter transfer function.

Return: The function returns the filter coefficients b and a.

2. butter_lowpass_filter(data, cutoff, fs, order=5):

This function applies the Butterworth low-pass filter defined in butter_lowpass to the input EDA signal data.

Arguments:

data: This is the raw EDA signal data as a 1D numpy array.

cutoff: Same as in butter_lowpass.

fs: Same as in butter_lowpass.

order: Same as in butter_lowpass (Default: 5).

Functionality:

Call butter_lowpass: It calls the butter_lowpass function with the provided arguments to obtain the filter coefficients b and a.

Filtering: It uses the scipy.signal.lfilter function to apply the designed filter to the input data. This function takes the filter coefficients (b and a) and the signal data as arguments.

Return: The function returns the filtered EDA signal data.

Overall:

These functions work together to achieve low-pass filtering on your EDA signal. The butter_lowpass function defines the filter characteristics, while butter_lowpass_filter applies that filter to the actual data.

By using a low-pass filter, you can remove high-frequency noise components from your EDA signal that might not be relevant to stress detection. This can improve the quality of the signal and potentially enhance the performance of stress detection algorithms.

share

more_vert

Can you walk me through some sample raw eda signal data samples and visualize these raw and transformed signals with matplotlib with pictures

edit

Show drafts

volume_up

Certainly! Here's a breakdown of sample raw EDA signal data, its visualization with matplotlib, and the transformation using the previously discussed Butterworth low-pass filter:

Sample Raw EDA Data:

EDA data typically consists of a series of numbers representing the skin's conductance over time. Here's an example of a short snippet:

Python

```
# Sample EDA data (simulated)
raw_eda_data = [0.2, 0.31, 0.28, 0.35, 0.42, 0.38, 0.4, 0.32, 0.29, 0.34, 0.41, 0.37]
```

Use code with caution.

content_copy

This is a very short sample, but it represents the basic structure of EDA data. The actual values can vary depending on the individual and the recording conditions.

Visualization with Matplotlib:

Python

```
import matplotlib.pyplot as plt

# Time axis (assuming sampling rate of 100 Hz)
time_axis = [i / 100 for i in range(len(raw_eda_data))]  # Time in seconds

# Plot the raw EDA data
plt.plot(time_axis, raw_eda_data, label="Raw EDA")
plt.xlabel("Time (s)")
plt.ylabel("Skin Conductance")
plt.title("Raw EDA Signal")
plt.legend()
plt.grid(True)
plt.show()
```

Use code with caution.

content_copy

This code snippet will generate a plot showing the raw EDA data over time. The x-axis represents time in seconds (assuming a sampling rate of 100 Hz), and the y-axis represents skin conductance.

Transformation with Butterworth Low-Pass Filter:

Python

```
from scipy.signal import butter, lfilter

# Define filter parameters (replace with appropriate values for your data)
cutoff_freq = 5  # Cut-off frequency in Hz
sampling_rate = 100  # Sampling rate in Hz
filter_order = 5  # Filter order

# Filter coefficients
nyq = 0.5 * sampling_rate
normal_cutoff = cutoff_freq / nyq
b, a = butter(filter_order, normal_cutoff, btype='lowpass', analog=False)

# Filter the data
filtered_data = lfilter(b, a, raw_eda_data)

# Plot the filtered EDA data (same as before with filtered data)
plt.plot(time_axis, filtered_data, label="Filtered EDA")
plt.xlabel("Time (s)")
plt.ylabel("Skin Conductance")
plt.title("Raw and Filtered EDA Signals")
plt.legend()
```

```
plt.grid(True)

plt.show()
```

Use code with caution.

content_copy

This code incorporates the butter and lfilter functions from scipy.signal to apply the Butterworth low-pass filter. You'll need to adjust the cutoff_freq, sampling_rate, and filter_order based on your specific data and desired filtering characteristics.

The resulting plot will show both the raw and filtered EDA signals. The filtered signal should have smoother variations, with high-frequency noise components removed.

Visualizing the Difference:

By comparing the raw and filtered EDA signals in the same plot, you can see how the filter affects the data. The raw signal might show sharp spikes and rapid fluctuations, while the filtered signal should exhibit a smoother trend, potentially highlighting the underlying physiological response better.

Important Note:

The provided sample data and filtering parameters are for illustrative purposes only. In a real scenario, you'll need to analyze your actual EDA data and determine appropriate filtering parameters based on its characteristics and noise content.