MATLAB implementation of iPPG signal processing

On average, it takes about 50-60 seconds to process 20 seconds long video. Exact time might vary depending on the selected region of interest, file size and PC specification.

It is a good practice to start your script with:

```
close all; clearvars;
```

This ensures all previous figures and variables in the Workspace are closed and cleared before repeating the analysis.

1. **Navigate to and read the file**

Browse and navigate to the file of interest. MATLAB stores the file name and path to this file separately. In order to find the absolute filename, those parameters should be combined into a single string of characters, `file_name`

Video file is then read by a special function `VideoReader` and is stored in a variable `mov`

2. **Get file properties**

Gets specific file properties

3. **Read one frame**

Read one frame from the video file, e.g. 50th frame. Create 3 copies of this frame into 3 variables (red, green, blue).

Decompose 3-channel RGB frame into mono-channel by setting any 2 channels to zero.

E.g., in order to get a red channel only, set 2nd and 3rd channel to zero.

4. **Plot original (RGB) and mono-channel frame**

Create a figure with 4 subplots:

```
subplot(2,2,p)
```

From MATLAB help:

`subplot(m,n,p)` divides the current figure into an `m`-by-`n` grid and creates an axes for a subplot in the position specified by `p`. MATLAB® numbers its subplots by row, such that the first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If the axes already exists, then the command `subplot(m,n,p)` makes the subplot in position `p` the current axes.

Plot each individual frame one-by-one by incrementing the variable `p`

5. **Region of interest**

Create a separate figure and plot previously selected frame. Draw rectangular and get its coordinates:

```
rect1 = getrect;
```

Round the coordinate values of the starting point and calculate the coordinate values of the end point. Draw rectangular to see its borders.

6. **Process frames**

Pre-allocate memory to speed-up the entire PPG computation. Three signals, one per mono-channel, are required. Set the length of these signals equal to the number of video frames.

Construct a big for-loop. Read one frame at a time:

```
current_frame = read(mov,k);
```

Read a specific portion of this frame using previously obtained coordinates. Calculate a single average value of all selected pixels and write it back to the memory as k-th element:

```
red_mean(k) = mean(red_ROI(:));
```

Repeat for all 3 channels.

7. **Plot raw PPG signals**

Plot raw PPG signals

8. **Filter raw signals**

Depending on the quality of the raw signals, a further filtering stage might be required. Set the low and high cut-off frequencies. Normalise with respect to Nyquist frequency.

Design Butterworth filter. From MATLAB help:

`[b,a] = butter(n,Wn,ftype)` designs a lowpass, highpass, bandpass, or bandstop Butterworth filter, depending on the value of `ftype` and the number of elements of `Wn`. The resulting bandpass and bandstop designs are of order `2n`.

Apply filter to the raw PPG signal. From MATLAB help:

`y = filtfilt(b,a,x)` performs zero-phase digital filtering by processing the input data, `x`, in both the forward and reverse directions [1]. `filtfilt` operates along the first nonsingleton dimension of `x`. The vector `b` provides the numerator coefficients of the filter and the vector `a` provides the denominator coefficients. If you use an all-pole filter, enter `1`for `b`. If you use an all-zero filter (FIR), enter `1` for `a`. After filtering the data in the forward direction, `filtfilt` reverses the filtered sequence and runs it back through the filter.

9. Plot filtered signals

Plot filtered PPG signals

10. Frequency analysis

Perform FFT and compute absolute values. Construct frequency axis. Plot frequency content for each PPG signal.

Note the Nyquist frequency and aliasing on the right-hand side of the frequency spectrum.

11. Calculate HR

Peak in the frequency spectrum plot should correspond to the HR in Hz. Find its position by calling `max` function. The result is not in Hz however, but rather in what is called “bins”. The number of bins equals the number of frames in the video; the longer the video, the more bins are in the frequency spectrum and greater the precision of the readings.

In order to convert the bin number into corresponding frequency, find its value in the frequency axis string `f_axis`.

Convert Hz into beat-per-minute (BPM) by multiplying this value by 60 seconds. You may also round the final reading if desired.