İçindekiler

Code, Graph, and Detection Explanation

Peak Detection of National Library Of Medicine

*Source -> **https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9280335/***

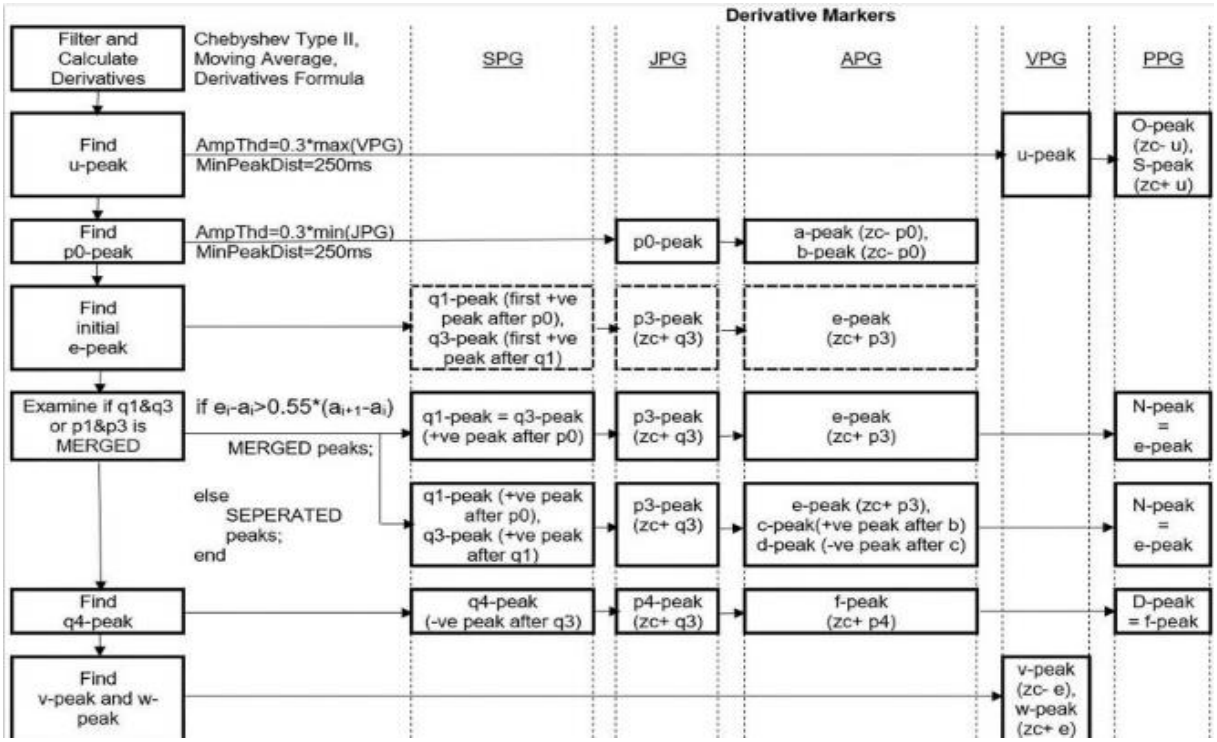
$$x[n] = \frac{1}{W}\left(y\left[n - \frac{W-1}{2}\right] + \ldots + y[n] + \ldots + y\left[n + \frac{W-1}{2}\right]\right) \quad (5)$$

The first vital peak to be detected is the u-peak of VPG. The peak is dominant and typically higher in amplitudes compared to the w-peak. An amplitude threshold of 0.3 * max (VPG) is set with a minimum peak distance of 250 ms since there is less likely for four pulses (four u-peaks) to appear in 1 s in rest condition (HR=240 bpm). The O-peak and S-peak of the PPG are determined by finding the zero-crossing point before and after the u-peak, respectively. Similar to u-peak, p0 of JPG is dominant in the negative amplitude. A threshold of 0.3 * min (JPG) with a minimum peak distance of 250 ms is used to get a negative p0-peak. The a-peak and b-peak of APG are then determined using the zero-crossing information of the p0-wave in the third derivative waveform.

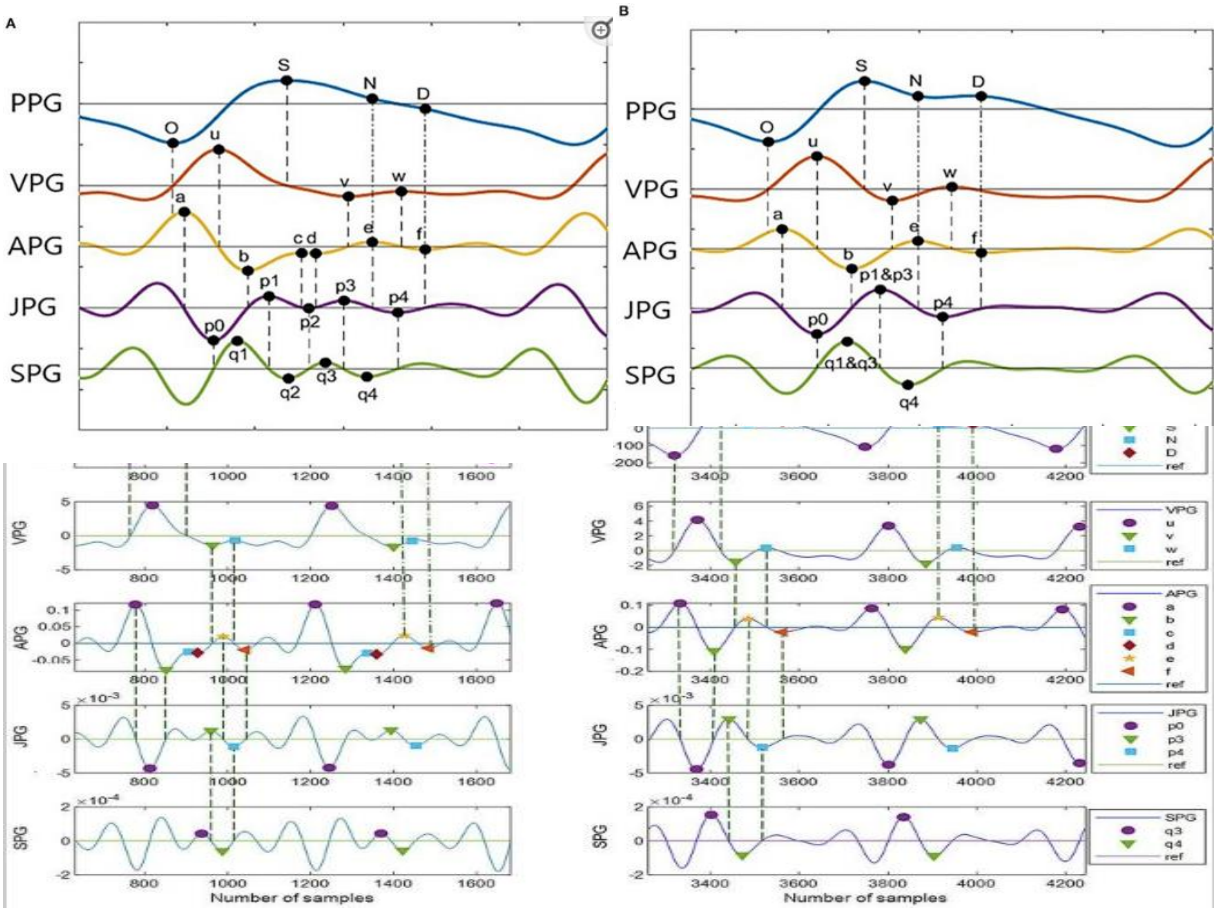
There are possibilities of merged peaks in APG, JPG, and SPG. The peaks of p1, p2, and p3 are better recognized using zero-crossing information of the SPG. Thus, the study proposed to determine q1-peak and q3-peak first. The q1-peak is the first positive peak after the p0-peak location. The second positive peak is q3-peak. Then, zero-crossing after q3-peak is p3-peak of JPG. Zero-crossing after the p3-peak is the e-peak. The e-peak then examines whether its location is bigger than 0.55 * current a-a interval. If yes, then the q1-q3 peaks and p1-p3 peaks are considered merged. The factor of 0.55 is chosen after a series of tests and trials on our dataset (subjects aged 32 years and above).

The q4-peak of SPG is determined afterward. It is the negative peak following the q3-peak. Zero-crossing after q4-peak is equal to p4-peak location, and zero-crossing after p4-peak is the f-peak in APG. Thus, the N-peak and D-peak of PPG are equal to the e-peak and f-peak of APG, respectively. There are cases where p2-wave has a positive value only. Thus, c-peak and d-peak cannot be determined using zero-crossing information. Therefore, the positive peak found between the b-peak and e-peak of APG is the c-peak, and the negative peak between the c-peak and e-peak is the d-peak. Meanwhile, the v-peak and w-peak of VPG are represented by zero-crossing before and after e-peak, respectively. The whole process is simplified as in Figure 4.

Peak Detection Explanation Graph



Note: the important part is finding initial e-peak and calculate the unclear (N-D) peaks and clear (N-D) peaks depending on e_i-a_i>0.55(a_{i+1}-a_i) formula*



Codes

Preprocessing Functions

```
import subprocess
import os
# get ready to use PPG signal
raw_signals = pd.read_csv(os.getcwd()+"/mimic_perform_af_csv/mimic_perform_af_001_data.csv")
ppg = raw_signals["PPG"].to_numpy()
# sampling frequency 360Hz
fs = 125

# generating time axis values
time = np.arange(ppg.size) / fs
winsize = fs * 4
winhop = fs * 2
i = 10

# Smoothing the PPG signal using a moving average filter
window_size = 10  # Adjust the window size as needed
ppg_smoothed = np.convolve(ppg, np.ones(window_size)/window_size, mode='same')

def compute_derivative(signal):
    return np.gradient(signal)
def butter_lowpass_filter(data, cutoff_frequency, fs, order=4):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff_frequency / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    y = filtfilt(b, a, data)
    return y


def baseline_correction(signal, window_size=101):
    # Apply a moving average to estimate the baseline
    baseline = np.convolve(signal, np.ones(window_size)/window_size, mode='same')

    # Subtract the baseline from the original signal
    corrected_signal = signal - baseline

    return corrected_signal
```

compute_derivative -> taking derivative of the signal

butter_lowpass_filter -> used for denoising

baseline_correction -> used to remove background effects

ZeroCrossing Functions

```python
def findZeroCrossingsFPP(signal,peaks):
    peak_indices = peaks

    zero_before_peak_indices = []
    zero_after_peak_indices = []

    for peak_index in peak_indices:

        zero_crossings_before_peak = np.where(np.diff(np.signbit(signal[:peak_index])))[0]


        if len(zero_crossings_before_peak) > 0:
            zero_before_peak_index = zero_crossings_before_peak[-1]
            zero_before_peak_indices.append(zero_before_peak_index)
        else:
            zero_before_peak_indices.append(0)

        zero_crossings_after_peak = np.where(np.diff(np.signbit(signal[peak_index:])))[0]


        if len(zero_crossings_after_peak) > 0:
            zero_after_peak_index = zero_crossings_after_peak[0] + peak_index
            zero_after_peak_indices.append(zero_after_peak_index)
        else:
            zero_after_peak_indices.append(len(signal)-1)
    return zero_before_peak_indices,zero_after_peak_indices

def findZeroCrossingsFNP(signal,peaks):
    peak_indices = peaks

    zero_before_negative_peak_indices = []
    zero_after_negative_peak_indices = []

    for peak_index in peak_indices:

        zero_crossings_before_peak = np.where(np.diff(np.signbit(-signal[:peak_index])))[0]


        if len(zero_crossings_before_peak) > 0:
            zero_before_peak_index = zero_crossings_before_peak[-1]
            zero_before_negative_peak_indices.append(zero_before_peak_index)
        else:
            zero_before_negative_peak_indices.append(0)


        zero_crossings_after_peak = np.where(np.diff(np.signbit(-signal[peak_index:])))[0]


        if len(zero_crossings_after_peak) > 0:
            zero_after_peak_index = zero_crossings_after_peak[0] + peak_index
            zero_after_negative_peak_indices.append(zero_after_peak_index)
        else:
            zero_after_negative_peak_indices.append(len(signal) - 1)

    return zero_before_negative_peak_indices,zero_after_negative_peak_indices
```

findZeroCrossingsFPP -> is used to find first zero crossings before and after positive directed peaks

findZeroCrossingsFNP -> is used to find first zero crossings before and after negative directed peaks

findEKeyPoints Function

```
def findEKeyPoints(SPG,JPG,jpg_p0_peaks,apg_a_peaks):
    peaks, _ = find_peaks(SPG)

    init_q1_peaks = []
    init_q3_peaks = []
    conc_q1_peaks = []
    conc_q3_peaks = []
    conc_q1_q3_peaks = []
    conc_e_peaks = []

    peaks_after_points = {}
    for point_index in jpg_p0_peaks:
        peaks_after_point = [peak for peak in peaks if peak > point_index]
        peaks_after_points[point_index] = peaks_after_point
        if(len(peaks_after_points[point_index]) > 0):
            init_q1_peaks.append(peaks_after_points[point_index][0])
        if(len(peaks_after_points[point_index]) > 1):
            init_q3_peaks.append(peaks_after_points[point_index][1])
    _,zero_amplitude_after_q3 = findZeroCrossingsFPP(SPG, init_q3_peaks)
    init_p3_peaks = zero_amplitude_after_q3
    _,zero_amplitude_after_p3 = findZeroCrossingsFPP(JPG, init_p3_peaks)
    init_e_peaks = zero_amplitude_after_p3


    peaks_after_points = {}
    for i in range(len(jpg_p0_peaks)):
        peaks_after_point = [peak for peak in peaks if peak > jpg_p0_peaks[i]]
        if(len(peaks_after_point) > 0):
            peaks_after_points[jpg_p0_peaks[i]] = peaks_after_point
            if i+1 in range(len(apg_a_peaks)):
                if(init_e_peaks[i]-apg_a_peaks[i] > 0.55*(apg_a_peaks[i+1]-apg_a_peaks[i])):
                    if(len(peaks_after_points[jpg_p0_peaks[i]]) > 0):
                        conc_q1_q3_peaks.append(peaks_after_points[jpg_p0_peaks[i]][0])
                else:
                    if(len(peaks_after_points[jpg_p0_peaks[i]]) > 0):

                        conc_q1_peaks.append(peaks_after_points[jpg_p0_peaks[i]][0])
                    if(len(peaks_after_points[jpg_p0_peaks[i]]) > 1):

                        conc_q3_peaks.append(peaks_after_points[jpg_p0_peaks[i]][1])
            else:
                conc_q1_q3_peaks.append(peaks_after_points[jpg_p0_peaks[i]][0])

    _,zero_amplitude_after_q1_q3 = findZeroCrossingsFPP(SPG, conc_q1_q3_peaks)
    _,zero_amplitude_after_q3 = findZeroCrossingsFPP(SPG, conc_q3_peaks)
    conc_separated_p3_peaks = zero_amplitude_after_q3
    conc_merged_p3_peaks = zero_amplitude_after_q1_q3
    _,temp_1_e_peaks = findZeroCrossingsFPP(JPG, conc_separated_p3_peaks)
    _,temp_2_e_peaks = findZeroCrossingsFPP(JPG, conc_merged_p3_peaks)
    conc_e_peaks = list(set(temp_1_e_peaks) | set(temp_2_e_peaks))
```

findEKeyPoints -> an algorithm that finds initial e, q3, and p3 peaks, then calculates the real position of e, q3, and p3 peaks depending on the formula and flowchart mentioned on peak detection section

findFKeyPoints Function

```
def findFKeyPoints(SPG,JPG,jpg_merged_p3_peaks,jpg_separated_p3_peaks,):
    spg_q4_peaks = []
    peaks,_ = find_peaks(-SPG)
    peaks_after_points = {}

    for point_index in jpg_merged_p3_peaks:
        peaks_after_point = [peak for peak in peaks if peak > point_index]
        peaks_after_points[point_index] = peaks_after_point
        if(len(peaks_after_points[point_index]) > 0):
            spg_q4_peaks.append(peaks_after_points[point_index][0])
    for point_index in jpg_separated_p3_peaks:
        peaks_after_point = [peak for peak in peaks if peak > point_index]
        peaks_after_points[point_index] = peaks_after_point
        if(len(peaks_after_points[point_index]) > 0):
            spg_q4_peaks.append(peaks_after_points[point_index][0])
    _,zero_amplitude_after_q4 = findZeroCrossingsFNP(SPG, spg_q4_peaks)
    jpg_p4_peaks = zero_amplitude_after_q4
    _,zero_amplitude_after_p4 = findZeroCrossingsFNP(JPG, jpg_p4_peaks)
    apg_f_peaks = zero_amplitude_after_p4
    ppg_distolic_peaks = apg_f_peaks

    return jpg_p4_peaks,apg_f_peaks,ppg_distolic_peaks,spg_q4_peaks
```

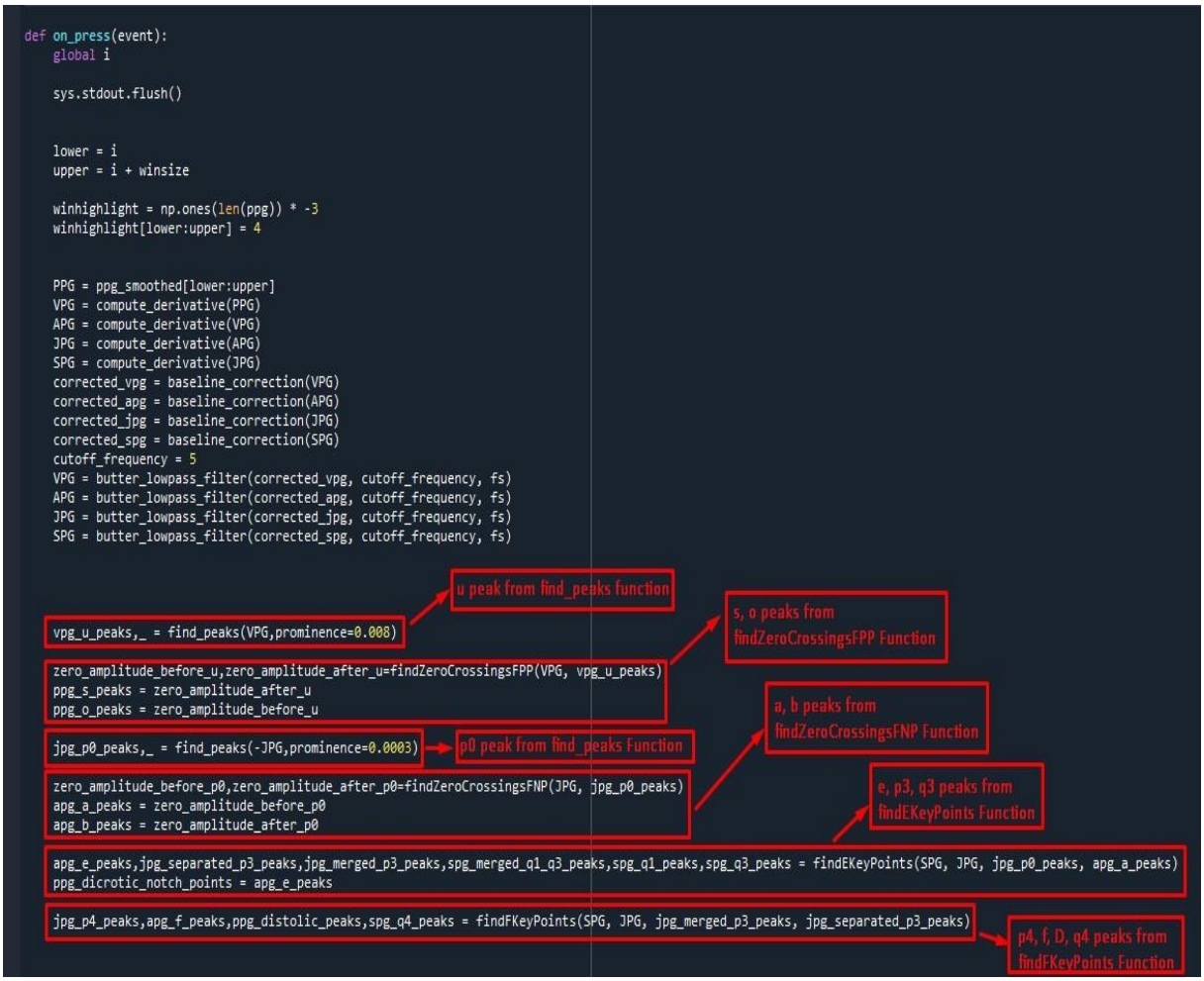
findFKeyPoints -> an algorithm that finds diastolic, q4, p4 and f peaks depending flowchart mentioned on peak detection section

PlottingTimeDifferencesOnPPG

```
def PlottingTimeDifferencesOnPPG(PPG,ppg_o_peaks,ppg_s_peaks,ppg_n_peaks,ppg_d_peaks,lower,upper):
    ppg_o_peaks = np.sort(ppg_o_peaks)
    ppg_s_peaks = np.sort(ppg_s_peaks)
    ppg_d_peaks = np.sort(ppg_d_peaks)
    ppg_n_peaks = np.sort(ppg_n_peaks)

    for i in range(len(ppg_o_peaks)):

        ax2.plot([ppg_o_peaks[i], ppg_o_peaks[i]], [PPG[ppg_o_peaks[i]], -0.4], 'k-')
        global_time_value = time[lower + ppg_o_peaks[i]]
        ax2.text(ppg_o_peaks[i]+5, PPG[ppg_o_peaks[i]] - 0.15, f'({global_time_value:.2f} s)', fontsize=7, color='black', ha='center', va='bottom')

        if(i < len(ppg_o_peaks)-1):
            #PWD
            time_diff_pwd = time[ppg_o_peaks[i+1]-ppg_o_peaks[i]]
            ax2.plot([ppg_o_peaks[i], ppg_o_peaks[i+1]],[-0.4,-0.4],'k-')
            ax2.text((ppg_o_peaks[i]+ppg_o_peaks[i+1])/2,-0.4, f'PWD = {time_diff_pwd:.2f} s', fontsize=6, color='black', ha='center', va='bottom')

            #Systolic Phase Diastolic Phase
            time_diff_sp = time[ppg_s_peaks[i]-ppg_o_peaks[i]]
            time_diff_dp = time[ppg_o_peaks[i+1]-ppg_s_peaks[i]]
            ax2.plot([ppg_s_peaks[i], ppg_s_peaks[i]], [PPG[ppg_s_peaks[i]], 0], 'k-')
            ax2.plot([ppg_o_peaks[i], ppg_s_peaks[i]],[-0,-0],'k-')
            ax2.text((ppg_s_peaks[i]+ppg_o_peaks[i])/2,-0.2, f'Systolic Phase = {time_diff_sp:.2f} s', fontsize=5.5, color='black', ha='center', va='bottom')
            ax2.plot([ppg_s_peaks[i], ppg_o_peaks[i+1]],[-0,-0],'k-')
            ax2.text((ppg_s_peaks[i]+ppg_o_peaks[i+1])/2,-0.2, f'Diastolic Phase = {time_diff_dp:.2f} s', fontsize=5.5, color='black', ha='center', va='bottom')
            global_time_value = time[lower + ppg_s_peaks[i]]
            ax2.text(ppg_s_peaks[i]-4.5, PPG[ppg_s_peaks[i]] - 0.2, f'({global_time_value:.2f} s)', fontsize=7, color='black', ha='center', va='bottom')

            #PPT
            time_diff_ppt = time[ppg_d_peaks[i]-ppg_s_peaks[i]]
            ax2.plot([ppg_s_peaks[i], ppg_s_peaks[i]], [PPG[ppg_s_peaks[i]], 0.9], 'k-')
            ax2.plot([ppg_d_peaks[i], ppg_d_peaks[i]], [PPG[ppg_d_peaks[i]], 0.9], 'k-')
            ax2.plot([ppg_s_peaks[i], ppg_d_peaks[i]],[0.9,0.9],'k-')
            ax2.text((ppg_s_peaks[i]+ppg_d_peaks[i])/2,0.75, f'PPT = {time_diff_ppt:.2f} s', fontsize=6, color='black', ha='center', va='bottom')

            #PWSP,PWDP,Dicrotich Notch
            ax2.plot([ppg_d_peaks[i], ppg_d_peaks[i]], [PPG[ppg_d_peaks[i]], 0], 'k-')
            ax2.plot([ppg_n_peaks[i], ppg_n_peaks[i]], [PPG[ppg_n_peaks[i]], 0], 'k-')
            amplitude_s = PPG[ppg_s_peaks[i]]
            amplitude_d = PPG[ppg_d_peaks[i]]
            amplitude_n = PPG[ppg_n_peaks[i]]
            ax2.text(ppg_s_peaks[i]+1.5, PPG[ppg_s_peaks[i]] /2 - 0.15 , f'{amplitude_s:.2f}', fontsize=6.5, color='black', ha='center', va='bottom',rotation=90)
            ax2.text(ppg_d_peaks[i]-1.2, PPG[ppg_d_peaks[i]] /2 - 0.12 , f'{amplitude_d:.2f}', fontsize=6.5, color='black', ha='center', va='bottom',rotation=90)
            ax2.text(ppg_n_peaks[i]-1.2, PPG[ppg_n_peaks[i]] /2 - 0.12 , f'{amplitude_n:.2f}', fontsize=6.5, color='black', ha='center', va='bottom',rotation=90)

            #PWA
            amplitude_pwa = PPG[ppg_s_peaks[i]]-PPG[ppg_o_peaks[i]]
            ax2.plot([ppg_o_peaks[  i], ppg_o_peaks[i]], [PPG[ppg_o_peaks[i]], PPG[ppg_s_peaks[i]]], 'k-')
            ax2.plot([ppg_s_peaks[i], ppg_o_peaks[i]], [PPG[ppg_s_peaks[i]], PPG[ppg_s_peaks[i]]], 'k-')
            ax2.plot([ppg_o_peaks[i], ppg_o_peaks[i]-3], [(PPG[ppg_s_peaks[i]]+PPG[ppg_o_peaks[i]])/2, PPG[ppg_s_peaks[i]]+0.2], 'k-')
            ax2.text(ppg_o_peaks[i]-3, PPG[ppg_s_peaks[i]]+0.22, f'PWA = {amplitude_pwa:.2f}', fontsize=6.5, color='black', ha='center', va='bottom')

            #Heart Rate ( BPM )
            ax2.plot([ppg_s_peaks[i], ppg_s_peaks[i]], [PPG[ppg_s_peaks[i]], 1.2], 'k-')
            if(i < len(ppg_o_peaks)-2):
                heart_rate = (fs/(ppg_s_peaks[i+1] - ppg_s_peaks[i])) * 60
                ax2.plot([ppg_s_peaks[i], ppg_s_peaks[i+1]],[1.2,1.2],'k-')
                ax2.text((ppg_s_peaks[i]+ppg_s_peaks[i+1])/2,1, f'HR = {heart_rate:.2f} s', fontsize=6, color='black', ha='center', va='bottom')
```

PlottingTimeDifferencesOnPPG -> an algorithm adds PWD, Systolic Phase, Diastolic Phase, PPT, PWSP, PWDP, Dicrotic Notch, PWA, and Heart Rate into the PPG signal graph that will be shown on the Graphs Section

Peak Detection Functions Used in on_press



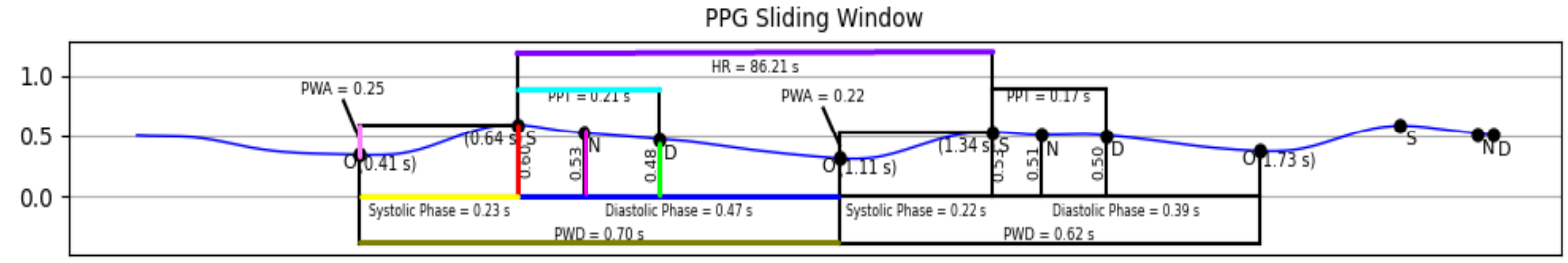
Plots in on_press

```
ax1.cla()
ax1.plot(time, ppg_smoothed, 'b', label='Smoothed PPG',linewidth = 0.75)  # Plot the smoothed signal
ax1.plot(time, winhighlight, 'r')
ax1.plot(time[lower:upper], ppg[lower:upper], 'r')
ax1.grid()
ax1.legend()
ax1.set_title('Raw Signal',fontsize=10)

ax2.cla()
ax2.plot(PPG, 'b',linewidth=1)
ax2.plot(ppg_s_peaks, PPG[ppg_s_peaks], 'ko',markersize = 5)
ax2.plot(ppg_o_peaks, PPG[ppg_o_peaks], 'ko',markersize = 5)
ax2.plot(ppg_dicrotic_notch_points,PPG[ppg_dicrotic_notch_points],'ko',markersize = 5)
ax2.plot(ppg_distolic_peaks,PPG[ppg_distolic_peaks],'ko',markersize = 5)
ax2.set_xticks([])
#ax2.set_ylim(-0.4,1.2)
##Adding PPG Points
for point in ppg_s_peaks:
    ax2.text(point+2, PPG[point] -0.2, 'S', fontsize=8, color='black', ha='center', va='bottom')

for point in ppg_o_peaks:
    ax2.text(point-2, PPG[point] - 0.15, 'O', fontsize=8, color='black', ha='center', va='bottom')

for point in ppg_dicrotic_notch_points:
    ax2.text(point+2, PPG[point]-  0.2, 'N', fontsize=8, color='black', ha='center', va='bottom')

for point in ppg_distolic_peaks:
    ax2.text(point+2, PPG[point] - 0.2, 'D', fontsize=8, color='black', ha='center', va='bottom')
##Plotting Time Differences
PlottingTimeDifferencesOnPPG(PPG, ppg_o_peaks,ppg_s_peaks,ppg_dicrotic_notch_points,ppg_distolic_peaks,lower,upper)


ax2.grid()
ax2.set_title('PPG Sliding Window',fontsize=10)

ax3.cla()
ax3.plot(VPG, 'r',linewidth=1)
ax3.plot(vpg_u_peaks,VPG[vpg_u_peaks],'ko')
for point in vpg_u_peaks:
    ax3.text(point, VPG[point] - 0.015, 'u', fontsize=12, color='black', ha='center', va='bottom')
ax3.set_xticks([])
ax3.grid()
ax3.set_title('VPG Sliding Window',fontsize=10)
```

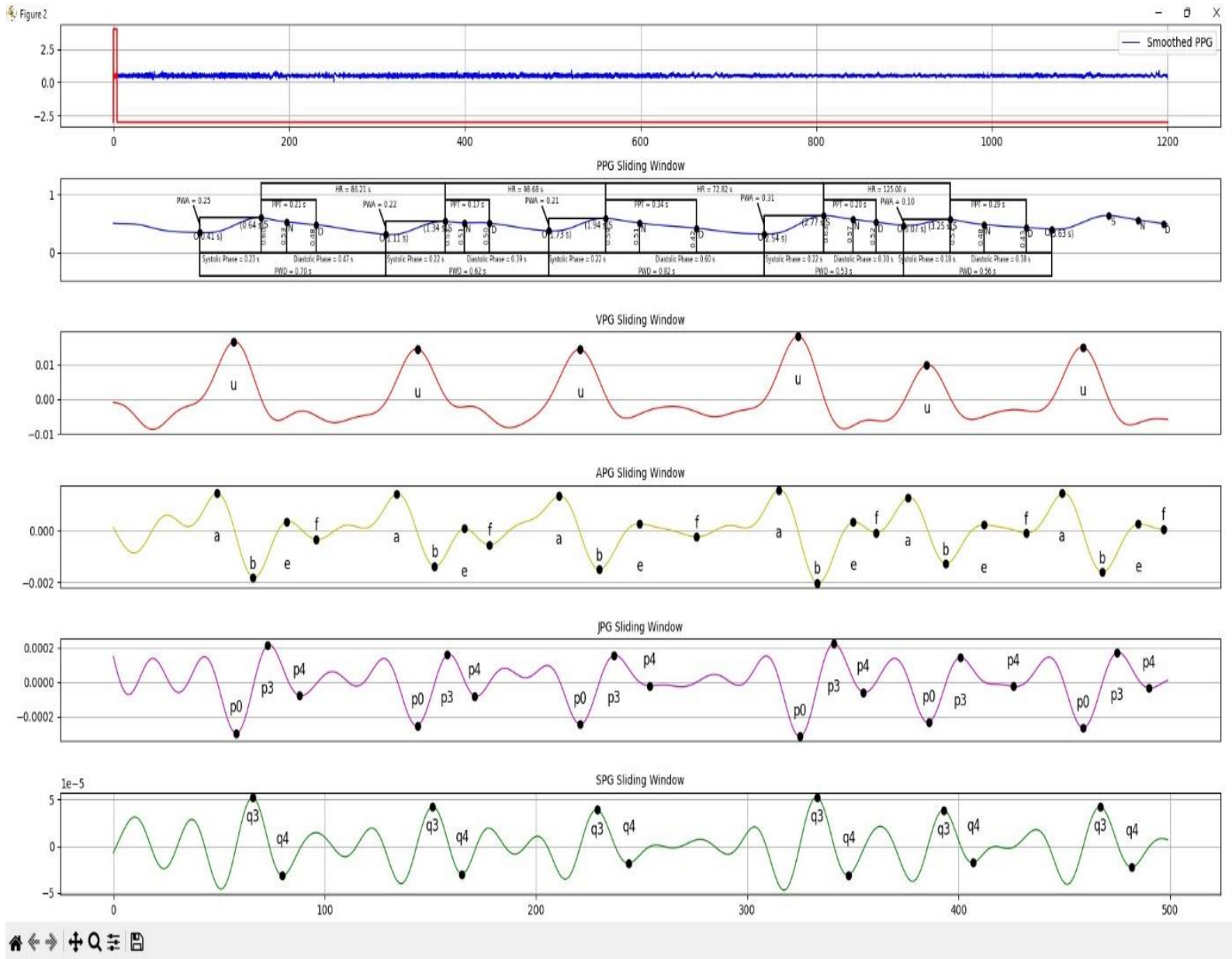
```
ax4.cla()
ax4.plot(APG,'y',linewidth=1)
ax4.plot(apg_a_peaks,APG[apg_a_peaks],'ko')
ax4.plot(apg_b_peaks,APG[apg_b_peaks],'ko')
ax4.plot(apg_e_peaks,APG[apg_e_peaks],'ko')
ax4.plot(apg_f_peaks,APG[apg_f_peaks],'ko')
for point in apg_a_peaks:
    ax4.text(point, APG[point] - 0.002, 'a', fontsize=12, color='black', ha='center', va='bottom')
for point in apg_b_peaks:
    ax4.text(point, APG[point] + 0.0002, 'b', fontsize=12, color='black', ha='center', va='bottom')
for point in apg_e_peaks:
    ax4.text(point, APG[point] - 0.002, 'e', fontsize=12, color='black', ha='center', va='bottom')
for point in apg_f_peaks:
    ax4.text(point, APG[point] + 0.0002, 'f', fontsize=12, color='black', ha='center', va='bottom')
ax4.set_xticks([])
ax4.grid()
ax4.set_title('APG Sliding Window',fontsize=10)

ax5.cla()
ax5.plot(JPG, 'm',linewidth=1)
ax5.plot(jpg_p0_peaks,JPG[jpg_p0_peaks], 'ko')
ax5.plot(jpg_separated_p3_peaks,JPG[jpg_separated_p3_peaks], 'ko')
ax5.plot(jpg_merged_p3_peaks,JPG[jpg_merged_p3_peaks],'ko')
ax5.plot(jpg_p4_peaks,JPG[jpg_p4_peaks],'ko')
ax5.set_xticks([])

for point in jpg_p0_peaks:
    ax5.text(point, JPG[point] + 0.0001, 'p0', fontsize=12, color='black', ha='center', va='bottom')
for point in jpg_separated_p3_peaks:
    ax5.text(point, JPG[point] - 0.0003, 'p3', fontsize=12, color='black', ha='center', va='bottom')
for point in jpg_merged_p3_peaks:
    ax5.text(point, JPG[point] - 0.0003, 'p3', fontsize=12, color='black', ha='center', va='bottom')
for point in jpg_p4_peaks:
    ax5.text(point, JPG[point] + 0.0001, 'p4', fontsize=12, color='black', ha='center', va='bottom')
ax5.grid()
ax5.set_title('JPG Sliding Window',fontsize=10)

ax6.cla()
ax6.plot(SPG, 'g',linewidth=1)
ax6.plot(spg_merged_q1_q3_peaks,SPG[spg_merged_q1_q3_peaks],'ko')
ax6.plot(spg_q3_peaks,SPG[spg_q3_peaks],'ko')
ax6.plot(spg_q4_peaks,SPG[spg_q4_peaks],'ko')
for point in spg_merged_q1_q3_peaks:
    ax6.text(point, SPG[point] - 0.00003, 'q3', fontsize=12, color='black', ha='center', va='bottom')
for point in spg_q3_peaks:
    ax6.text(point, SPG[point] - 0.00003, 'q3', fontsize=12, color='black', ha='center', va='bottom')
for point in spg_q4_peaks:
    ax6.text(point, SPG[point] + 0.00003, 'q4', fontsize=12, color='black', ha='center', va='bottom')

ax6.grid()
ax6.set_title('SPG Sliding Window',fontsize=10)
```

Graphs

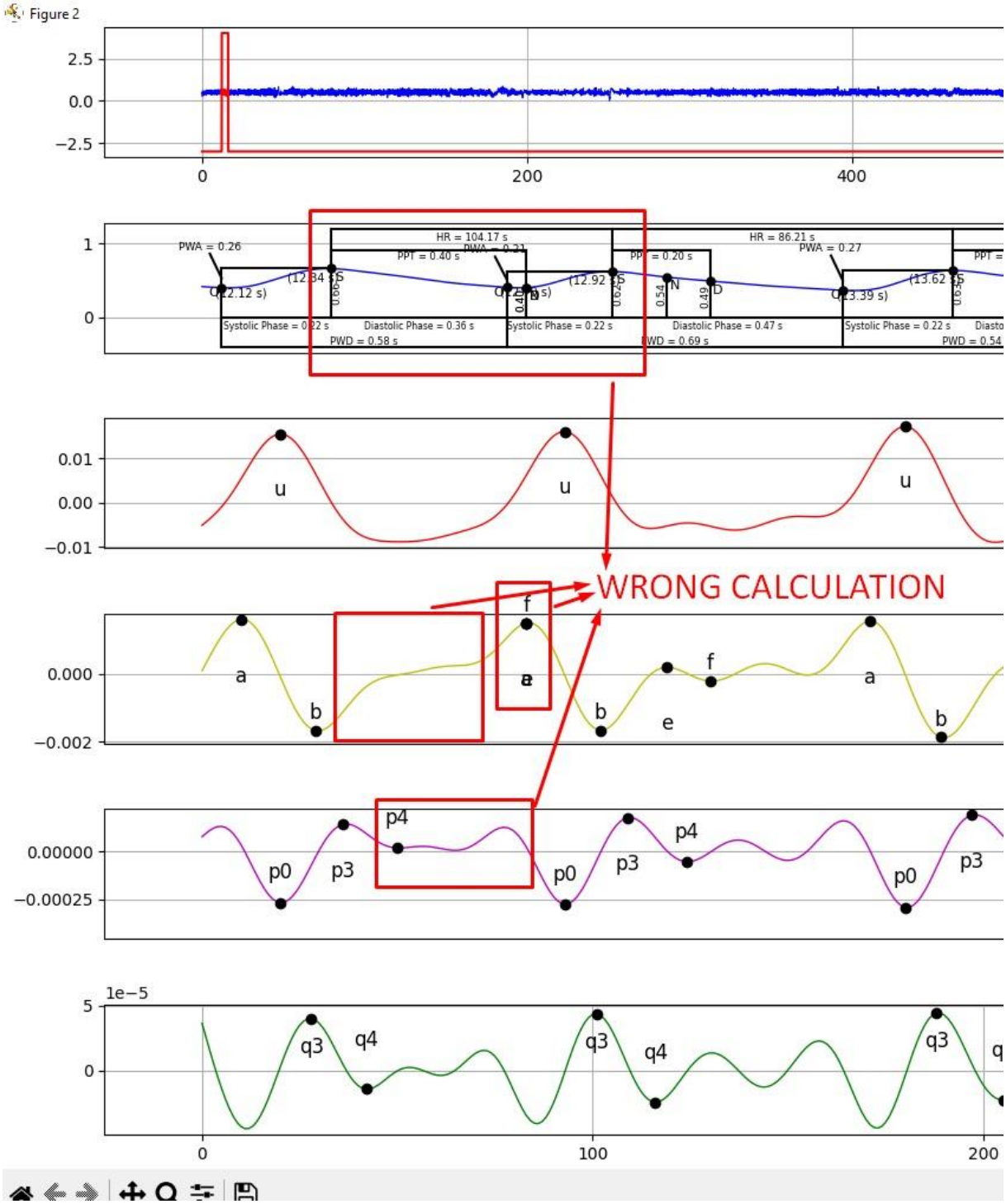
2 Cycle (O Keypoint to O Keypoint)



Graph shown current windowsize (winsize = fs*4)



Wrong Calculations



Note: Sometimes, the program is not working perfect and the wrong calculations appear in the results as shown in the graph above..