تمرینات کامپیوتری درس BSP

Question 1.1

The data file *'ecg2x60.dat'* contains ECG signal, sampled at 200Hz, with a significant amount of 60Hz power-line artifact.

a. Design a notch filter with two zeros to remove the artifact and implement it in MATLAB.

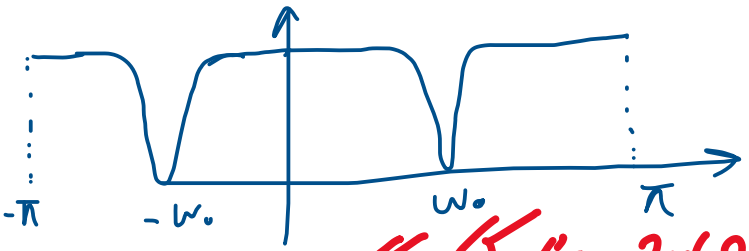
b. Add two poles at same frequency as those of zeros, but with a radius that is less than unity. Study the effect of poles on output of the filter as their radius is varied from 0.8 to 0.99.

c. Find the signal-to-noise ratio (SNR) for the above cases considering the best filter output as a reference signal

ترجمه: فایل "ecg2x60.dat" موجود در فایل زیپ، حاوی یک سیگنال ECG است که با فرکانس نمونه‌برداری 200Hz ایجاد شده است. این سیگنال حاوی نویز برق شهر است که اثر خودش را در فرکانس 60Hz بر روی سیگنال ECG گذاشته است.

الف) یک فیلتر ناچ (notch یا میان‌نگذر) طراحی کنید که می‌تواند این محتوای نویزی در فرکانس 60Hz را حذف کند.

راهنمایی: شکل فیلتر ناچ به شکل روبرو است:

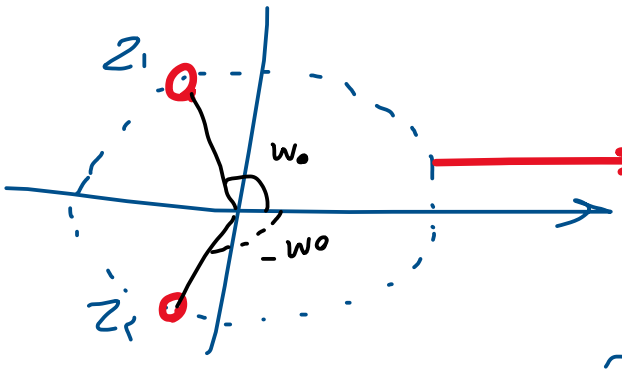


برای خواندن سیگنال می‌توانید از فایل متلب "ecg2x60.m" کمک بگیرید.

ادامه راهنمایی: شما باید w_0 را براساس فرکانس نمونه برداری و فرکانس نوتر برق شهر بدست آورید.

دقت کنید که $\pi \longrightarrow \frac{f_s}{2}$ تبدیل شده است.

شما باید یک فیلتر FIR طراحی کنید که دو صفر در موقعیتهای زیر داشته باشد.



به عبارت دیگر صفرهای شما

$z_1 = 1 \times e^{jw_0}$

و $z_2 = 1 \times e^{-jw_0}$

هستند.

فیلتر FIR شما به فرم زیر خواهد بود:

$$H(z) = (1 - z^{-1} z_1)(1 - z^{-1} z_2)$$

برای اینکه این فیلتر همی $H(z)$ برای باند گذر شود ($|H(z)| = 1$) به ازای بدست آوردن معادلات فیلتر $H(z)$ و آن را برحسب w تقسیم کنید که $|H(z)| = 1$ شود. **این عدد را شما باید بدست آورید.**

$H(z)$ که طراحی شده آن را در قالب به فرم کلی زیر کلست :

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_n z^{-n}}{a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}}$$

b_i ها را در یک بردار مانند b بگیرید $b = [b_0, b_1, \ldots]$

a_i را در یک بردار دیگر مانند a بگیرید $a = [a_0, a_1, \ldots]$

نکته مهم: بردار a و b باید هم طول باشند. اگر دیدید که تعداد b_i ها از a_i ها بیشتر است داخل بردار a بجای a_i های که وجود ندارند صفر بگذارید تا بردار a هم طول بردار b شود.

راههای مهم: از دستور tf در صلب برای ساخت فیلتر مورد نظرتان استفاده کنید. $\Leftarrow H = tf(b, a, \frac{1}{f_s})$ ($f_s = 2\ldots$)

از دستور "pzplot" برای نمایش فیلترتان استفاده کنید.

از دستور "filter" برای فیلترکردن سیگنالتان استفاده کنید.

$y = filter(b, a, x)$ $\Leftarrow$ (ورودی x سیگنال شما باشد)

y سیگنال خروجی شما خواهد شد.

از دستور "freqz" و "phasez" برای نمایش پاسخ اندازه و فاز $H(z)$ استفاده کنید.

سیگنال فیلتر شده را نمایش دهید. (چی می بینید؟ توضیح دهید)

ب) دو قطب به $H(z)$ بخش قبل در همان مکان زوایای $\pm\omega_0$ اضافه کنید ولی اندازه قطب‌ها را کمی از یک بگیرید. حال با فیلتر جدید، سیگنال خود را فیلتر کنید چه تغییری مشاهده می‌کنید؟ اندازه قطب‌ها را از ۰/۸ تا ۰/۹۹ با گام ۰/۰۵ تغییر دهید و نتایج فیلتر کردن را با تغییر اندازه قطب‌ها در شکل‌های مختلف نشان دهید.

راهنمایی: قطب‌های شما به شکل‌های $p_1 = r_1 e^{j\omega_0}$ و $p_2 = r_1 e^{-j\omega_0}$ خواهند بود که $0.8 < r_1 < 0.99$ خواهد بود.

$$H(z) = \frac{(1 - z^{-1}z_1)(1 - z^{-1}z_2)}{(1 - z^{-1}p_1)(1 - z^{-1}p_2)}$$

جدید خواهد بود. (یادتان نرود که $H(z)$ را نرمالیزه کنید که $|H(z)| = 1$ بشود.

ج) با امتحان تمامی فیلترهای بخش ب و SNR های آن‌ها را مقایسه کنید.

سیگنال مرجع را برای محاسبه‌ی SNR، خروجی فیلتر ناچ با قطب‌های $p_{1,2} = 0.99 e^{\pm j\omega_0}$ در نظر بگیرید.

راهنمایی: فرمول محاسبه‌ی SNR:

$$SNR = 10 \log_{10} \left(\frac{\text{Signal energy}}{\text{noise energy}} \right)$$

در معادله بالا Signal energy انرژی خروجی فیلتر ناچ است.

noise energy انرژی نویز است.

$$\text{noise} = \text{reference signal} - \eta$$

η ← ECG ورودی حاوی نویز

reference signal ← سیگنال مرجع

سیگنال مرجع در این مسئله همان خروجی فیلتر ناچ با قطب‌های $p_{1,2} = 0.99 e^{\pm j\omega_0}$ است.

نحوه محاسبه انرژی یک سیگنال گسسته مانند $\eta(n)$ به صورت زیر است:

$$\text{energy} = \sum_{i=1}^{n} |\eta(i)|^2$$

در متلب کد می‌نویسی به صورت زیر است: energy سیگنال x

```
energy = sum(x.^2);
```

Question 1.2

- Filter the noisy ECG signal in *'ecg_hfn.dat'* (sampling frequency=1000Hz) using four different Butterworth low pass filters (individually) realized through MATLAB with the following characteristics:
  (a)Order 2, cutoff frequency 10Hz;
  (b)Order 8, cutoff frequency 20Hz;
  (c)Order 8, cutoff frequency 40Hz;
  (d)Order 8, cutoff frequency 70Hz;

ترجمه: سیگنال موجود در فایل "ecg_hfn.dat" یک سیگنال ECG است که با فرکانس نمونه‌برداری ۱۰۰۰Hz ثبت شده است. برای خواندن فایل می‌توانید از فایل "ecg_hfn.m" که در متلب نوشته شده است کمک بگیرید. این سیگنال حاوی نویز فرکانس بالاست می‌خواهیم این سیگنال را با استفاده از فیلتر باترورث پایین‌گذر در مرتبه‌های مختلف فیلتر کنیم: خروجی فیلترهای زیر را رسم کنید

الف) باترورث مرتبه ۲ با فرکانس قطع ۱۰Hz

ب) " " ۸ " " ۲۰Hz

ج) " " ۸ " " ۴۰Hz

د) " " " " " ۷۰Hz

ه) SNR خروجی فیلترهای بالا را محاسبه کنید. سیگنال مرجع همان خروجی فیلتر ج. است.

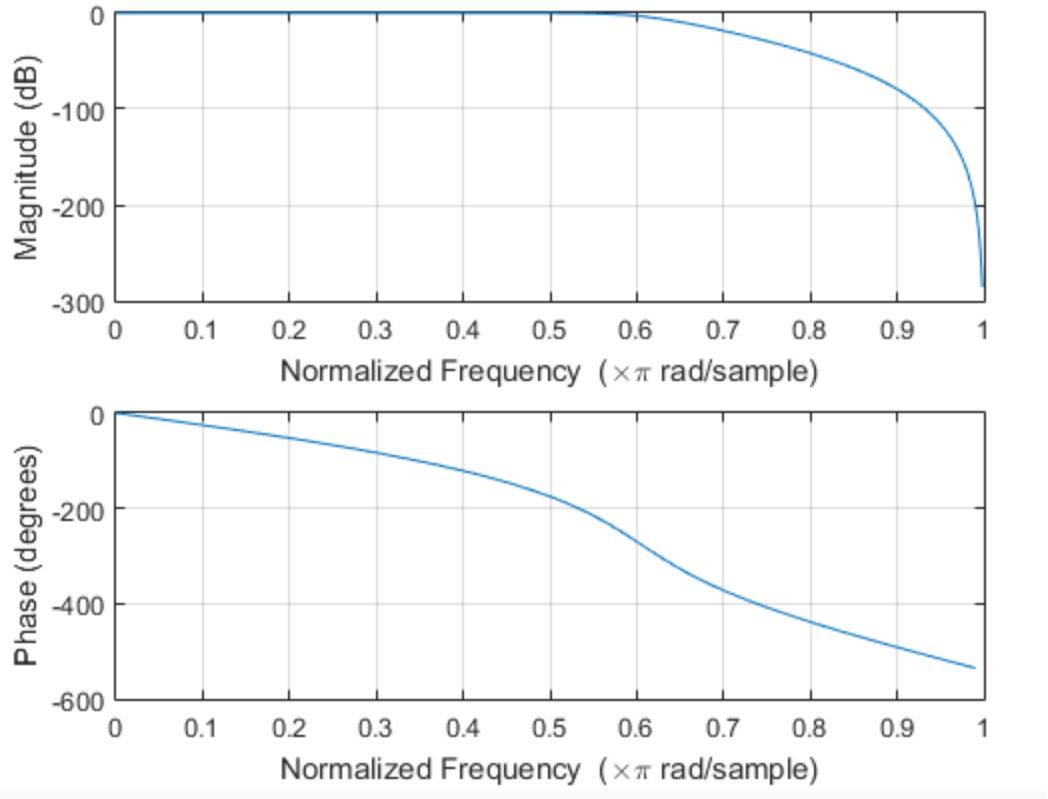
راهنمایی: از دستور "butter" در متلب برای ایجاد فیلتر باترورث استفاده کنید.

دقت کنید که باید فرکانس قطع را بر اساس فرکانس نمونه برداری نرمالیزه کنید.

به مثال زیر متلب توجه کنید:

Design a 6th-order lowpass Butterworth filter with a cutoff frequency of 300 Hz, which, for data sampled at 1000 Hz, corresponds to 0.6π rad/sample. Plot its magnitude and phase responses. Use it to filter a 1000-sample random signal.

```
[b,a] = butter(6,0.6);
freqz(b,a)
dataIn = randn(1000,1);
dataOut = filter(b,a,dataIn);
```

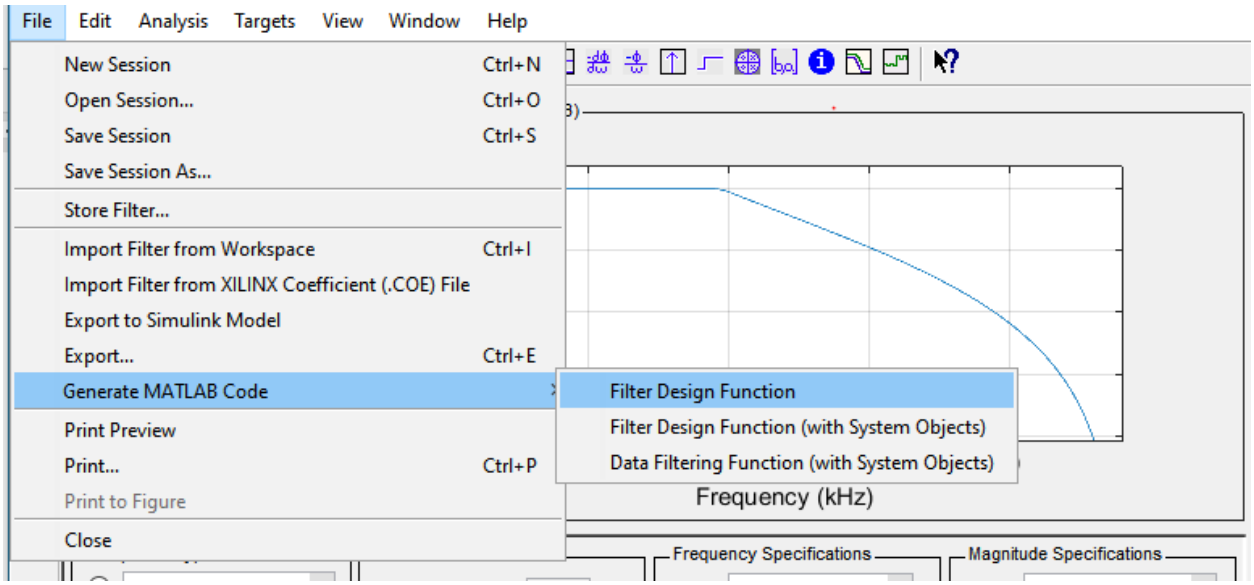


توجه کنید که در سوال بالا چیزی در مورد تضعیف باند عبور (A_p) و تضعیف باند توقف (A_s) به ما نگفته بودند چون از قبل مرتبه فیلتر را برای ما مشخص کرده بودند. اگر یادتان باشد در درس پردازش سیگنال گفته بودم برای طراحی فیلتر می‌توانید از دستور "fdatool" در متلب استفاده کنید.

۹) با معیارهای که در درس گفته شد یک فیلتر بازورت با حداکثر تضعیف باند عبور ۱dB و حداقل تضعیف باند توقف ۶۰dB و فرکانس قطع ۲۰Hz (فرکانس نمونه برداری ۱۰۰Hz) ایجاد کنید و سیگنال را فیلتر کنید و SNR آن را محاسبه کنید.

راهنمایی: چون در اینجا به شما As و Ap را گفته‌ام، در بخش fdatool در بخش filter order، گزینه minimum order را بجای specify order انتخاب کنید. اگر به شما مرتبه فیلتر را گفتیم از گزینه specify order استفاده کنید و مرتبه فیلتر را در آنجا وارد کنید.

بعد از طراحی فیلتر، باتوجه به عکس نشان داده شده زیر، از منوی فیلتر طراحی شده خود را تولید کنید و آن را ذخیره کنید.



کدهای داخل کادر را به غیر از خط اول در یک فایل جدید کپی کنید.
متغیر Hd در این کد همان فیلتر طراحی شده شماست با استفاده از دستورزیر سیگنال ECG خود را فیلتر کنید:

ecg_out = filter(Hd, n)

سیگنال ecg خروجی

سیگنال ecg ورودی

ز) با استفاده از fdatool این بار یک فیلتر چبی شف نوع ۱ و یک فیلتر چبی شف نوع ۲ با همان مشخصات زدایش قطع و باند تضعیف و بهبود سؤال قبلی این دو نیز خروجی آنها را با خروجی متلب قبل مقایسه کنید و SNR آنها را محاسبه کنید.

Question 1.3

- Filter the signal in the file *'ecg_lfn.dat'* (sampled at 1000Hz) (sampling frequency=1000Hz) using Butterworth high pass filters with order 2-8 and cut-off frequencies 0.5-5 Hz. Study the efficacy of filtering in removing the base-line drift and the effect on the ECG waveform itself. Determine the best compromise acceptable.

سیگنال ecg موجود در فایل "ecg_lfn.dat" یک سیگنال است که با فرکانس نمونه برداری 1000Hz گرفته شده است. از فایل متلب "ecg_lfn.m" برای فراخوانی این سیگنال استفاده کنید.
این سیگنال حاوی نویز فرکانس پایین انحراف خط مبناست.
با استفاده از فیلتر باترورث بالاگذر این انحراف خط مبنا را حذف کنید.
مرتبه فیلتر را از ۲ تا ۸ تغییر دهید. همچنین فرکانس قطع را از ۰/۵ تا 5Hz تغییر دهید و ببینید کدام فیلتر نتیجه بهتری می دهد.
خروجی همه فیلترها را نمایش دهید.

فیلترهایی نتیجه بهتری می دهد که با وجود کم کردن انرژی، شکل ecg کمترین آسیب رسانده باشد.

به جای فیلتر بالاگذر، از یک فیلتر متوقف‌کننده برای حذف نویز انحراف خط مبنا استفاده کنید (می‌دانیم فیلتر متوقف‌کننده فیلتر بالاگذر است)

حال مقدار فیلتر متوقف‌کننده را در فرکانس $f_c = 0.1\,Hz$ قرار دهید و نتایج را تحلیل کنید.

راهنمایی: یک فیلتر ساده متوقف‌کننده به صورت زیر است:

$$H(z) = (1 - z^{-1} z_1)$$

که در آن z_1 صفر این فیلتر است که حل آن $z_1 = 1 \times e^{+j\omega_c}$ است.

که در آن $\omega_c = 2\pi \frac{f_c}{f_s}$ است که f_c همان فرکانس قطع است.

بقیه راهنمایی‌ها را لازم در صورت ابهام فیلتر در سؤال‌های قبلی نیز نشان است.

فیلتر متوقف‌کننده بالا با وجود اینکه می‌تواند انحراف خط مبنا را حذف کند ولی به علت نزدیک قطب در نزدیکی صفر خود باعث خراب شدن محتوای سیگنال و هم‌پوشانی شده است. یک قطب به فیلتر بالا در همان فرکانس $f_c = 0.1$ متناسب با اندازه کوچکتر اضافه کنید

و به ازای اندازه های ۰٫۷ ، ۰٫۸ ، ۰٫۹ و ۰٫۹۹ = r

برای قطب فیلتر ها خروجی فیلتر را رسم و نتایج را آنالیز کنید.

راهنمایی : فیلتر شما به این شکل خواهد بود:

$$H(z) = \frac{(1 - z^{-1} z_1)}{(1 - z^{-1} p_1)}$$

$$p_1 = r\, e^{jw_c} \qquad r = ۰٫۷ ,\ ۰٫۸ ,\ ۰٫۹ ,\ ۰٫۹۹$$

بقیه راهنمایی ها در سوالهای قبل آمده است .

سوال 4.1: در این سوال می‌خواهیم تأثیر میانگین گیری سنکرون را بر روی یک سیگنال ecg مشاهده کنیم. سیگنال مورد فایل "ecg-hfn.dat" را بخوانید. این سیگنال حاوی یک سیگنال ecg آلوده به نویز فرکانس بالاست. فرکانس نمونه‌برداری این سیگنال 1... Hz است. یک فیلتر کامل از این سیگنال ecg را انتخاب می‌کنیم. (برای راحتی کار از نمونه n=950 تا نمونه n=980+801 را به عنوان یک فیلتر template انتخاب کنید) حال کاری که خواهید کرد این است که ضریب هم بستگی این سیگنال و فیلتر را با کل سیگنال ecg محاسبه کنید و خروجی را نمایش دهید. برای این کار باید از اول سیگنال ecg اصلی خود شروع به حرکت کرده و قطعاتی با طول سیگنال template، از سیگنال ecg استخراج کنید.

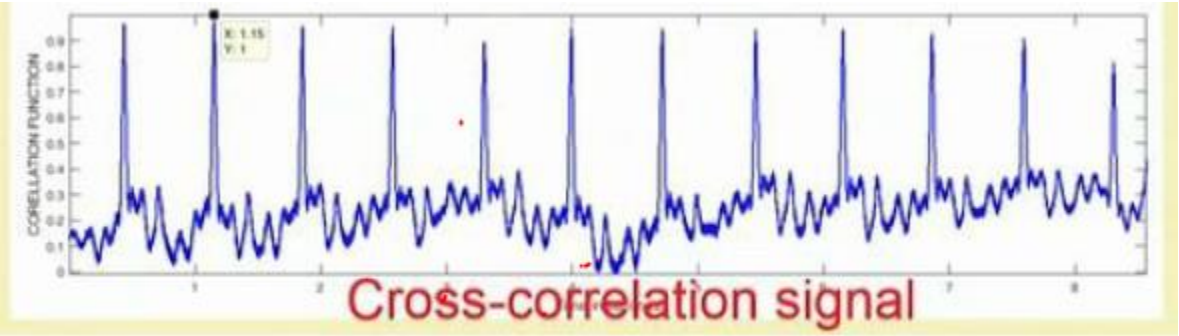
(واضح است که طول این قطعات ۸۰۲ نمونه خواهد شد). پس
با استفاده از دستور corrcoeff ضریب هم بستگی
قطعه استخراج شده با سیگنال template را محاسبه کنید. پس از محاسبه
و ذخیره مقدار ضریب هم بستگی، یک گام رو به جلو حرکت کنید و دوباره
قطعه ای از سیگنال ecg جدا کنید و مراحل بالا را تکرار کنید.
دوباره یک گام به جلو رفته و مراحل بالا تکرار کنید و اینقدر ادامه دهید
که به آخرین نمونه سیگنال ecg برسید:

راهنمایی: فرض کنید x سیگنال ecg و y همان template شما باشد.
با استفاده از دستورات زیر می توانید بردار ضریب هم بستگی را بدست آورید:

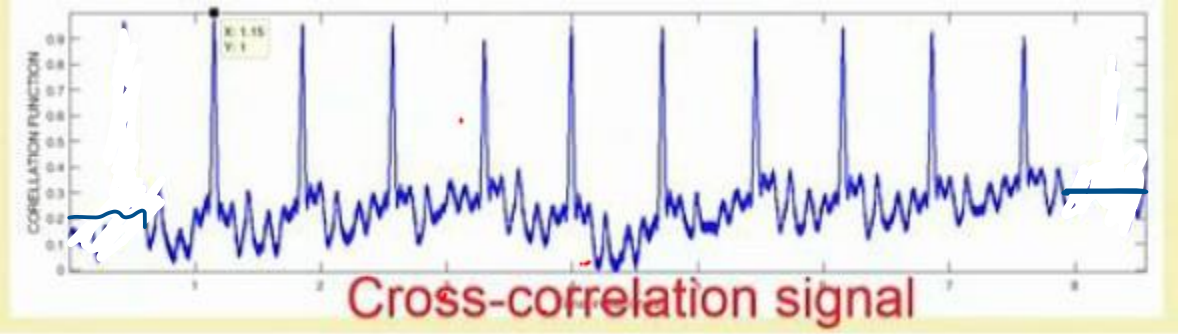
```
P = zeros(size(x));        ← بردار ضریب هم بستگی را با اندازه طول سیگنال ecg در نظر می گیریم.
Ly = length(y);            ← طول template
Lx = length(x);            ← طول ecg
x = [zeros(1, floor(Ly/2)); x; zeros(1, floor(Ly/2))];
for i = 1 : Lx
segment = x(i : i+Ly-1);
P(i) = corrcoeff(segment, y, 'coeff');
end
```

کار جالبی که قبل از حلقه for انجام دادیم zero padding یا صفرگذاری نام دارد. با این کار طول سیگنال ecg را با کاربرد صفرهای در ابتدا و انتها افزایش دادیم. تعداد صفرها باندازه طول template ما بود. zeropadding به این خاطر انجام می‌شود که ما در هنگام هم‌بستگی گرفتن و رفتن به جلو روی سیگنال ecg اصلی، از نمونه‌های انتهای سیگنال ecg هم نباید هم‌بستگی بگیریم. zeropadding کاربردهای زیادی در کانولوشن و غیره دارد.

سیگنال P ما اینگونه به شکل زیر خواهد بود:



اگر ما از zeropadding استفاده نمی‌کردیم سیگنال P ما به این شکل در می‌آمد. که اطلاعات ناقصی داشت. (مانند قله در اول و آخر را نداشتیم)



با استفاده از آستانه گذاری محل قله ها را پیدا کنیم.
آستانه را بزرگتر از ۰.۸ قله ها و محل قله ها را پیدا کنیم.
بهترین محل قله ها جاهایی است که template ما بیشترین هم بستگی را با سیگنال ورودی داشته است. یعنی ما با کدام فرمان شبیه به template مان در نقطه قله روزگر دهیم. آن نقطه ها را انتخاب کنیم و در یک ماتریس بریزیم. و سپس از ماتریس به صورت سطری میانگین گیری کنیم و خروجی آن را مشاهده کنیم. خروجی سیگنال تک میانگین گیری کردن از فرمان ها است

راهنمایی: با استفاده از دستور find می‌توانید تمام نمونه‌هایی که در P بزرگتر از ۰/۸ هستند را پیدا کنید:

```
idx = find( P > .8);
```

قطعه اولی که شبیه template ماست:

```
seg1 = x( idx(1) : idx(1) + Ly - 1)
```

قطعه ۵ ای که شبیه template ماست:

```
seg5 = x( idx(5) : idx(5) + Ly - 1)
```

می‌توانید با یک حلقه for این قطعات را استخراج کنید و در یک ماتریس بریزید: (چون قطعات ما به صورت سطری هستند ماتریس را اینگونه پر کنید:

```
matrix = [ ];        % این ماتریس آرایه است که در نهایت با قطعات ما پر خواهد شد
for i = 1 : length(idx)
    seg = x( idx(i) : idx(i) + Ly - 1);
    matrix = [matrix ; seg];        % این خط را همیشه بیاد داشته باشید
end
```

با استفاده از دستور mean از ماتریس میانگین بگیرید و خروجی را نشان دهید.

ب) با استفاده از دستور mean از ۴ سطر اول و سپس از ۷ سطر اول ماتریس میانگین بگیرید و با نتیجه اصلی مقایسه کنید.

ج) مقدار آستانه را ۹۵، ۹۰، و ۷۰ انتخاب کرده، و نتایج بالا را تکرار کنید و مقایسه کنید.

سؤالات اینجا تمام شده است و بقیه فایل به تمام چیزهای تمرین حل شده را نشان می‌دهد.

به شما توصیه می‌کنم حتماً این تمرین‌های حل شده را نگاه کنید. نکات خوبی دارند.

Question 2.3

- Design a Wiener filter to remove the artifacts in the ECG signal in the file *'ecg_hfn.dat'* (sampled at 1000Hz). The equation of desired filter is :

$$W(\omega) = \frac{S_d(\omega)}{S_d(\omega) + S_m(\omega)} = \frac{1}{1 + \frac{S_m(\omega)}{S_d(\omega)}}$$

where S_d and S_m represent the PSDs of desired signal and noise respectively.

Question 2.3 (*contd…*)

The required model PSD may be obtained as follows:

Create a piece-wise linear model of desired version of the signal by concatenating linear segments to provide P, QRS, and T waves with amplitudes, durations, and intervals similar to those in the given noisy ECG signal. Compute the PSD of the model signal. Select a few segments from the ECG signal that are expected to be iso-electric (for example T-P interval). Compute the PSD and obtain their average. The selected noise segment should be zero mean. Compare the results of Wiener filter with those obtained by synchronized averaging and low pass filtering.

Solution 2.3 *Cont.…*

- Input ECG signal is available at:
  http://people.ucalgary.ca/~ranga/enel563/SIGNAL_DATA_FILES/ecg_hfn.dat
- Sample MATLAB code to display the input ECG is available at:
  http://people.ucalgary.ca/~ranga/enel563/SIGNAL_DATA_FILES/ecg_hfn.m

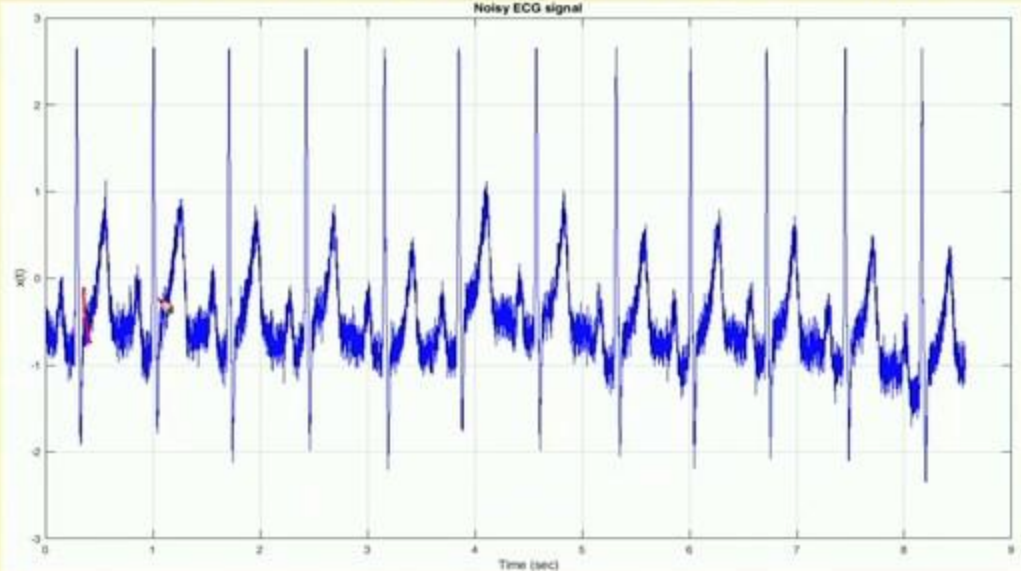
Note: Keep the input signal and the MATLAB codes in the same directory.

Solution 2.3

- Wiener filtering is the statistical approach to reduce noise in signal
- The Wiener filter minimizes the mean square error between the estimated random process and the desired process
- Wiener filter requires the assumption that the signal and noise processes are weak-sense stationary
- PSD of the desired signal and noise is required for Wiener filtering

Solution 2.3

- The input Signal with High frequency noise

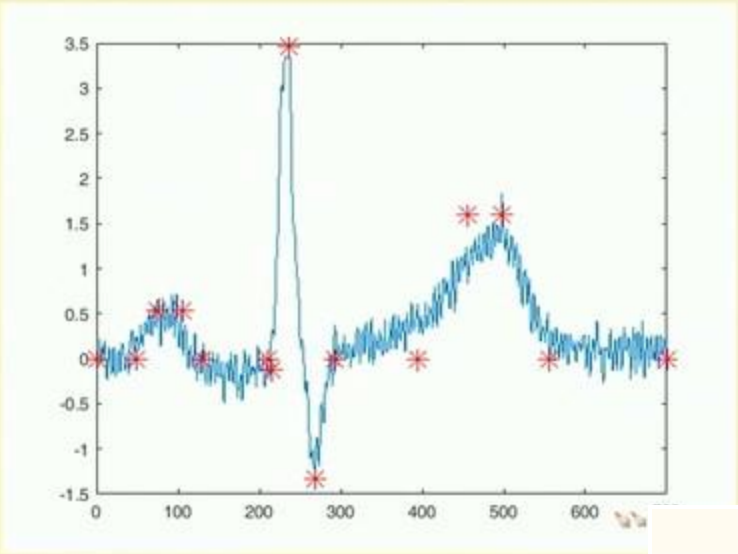



```
%% Load and Display Signal
% load Input ECG Signal
ecg = load('ecg_hfn.dat');
fs = 1000; % sampling freq
L =length(ecg);%Signal length
t=[1:L]/fs;
figure;
plot(t,ecg);
```

Solution 2.3

- Select points on one cycle of ECG for constructing piecewise linear model

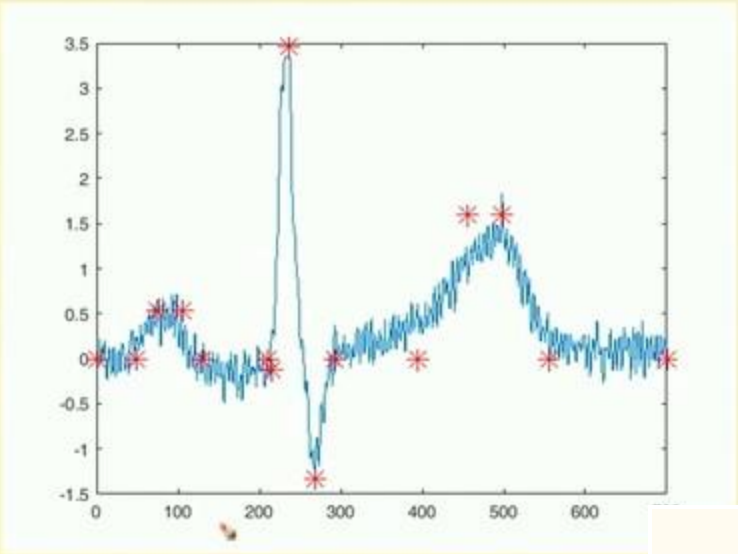
- Red points indicates endpoints of piecewise linear segments of ECG



Solution 2.3

- Select points on one cycle of ECG for constructing piecewise linear model

- Red points indicates endpoints of piecewise linear segments of ECG



Solution 2.3

- The desired signal is constructed by concatenating linear segments to provide P, QRS, and T waves

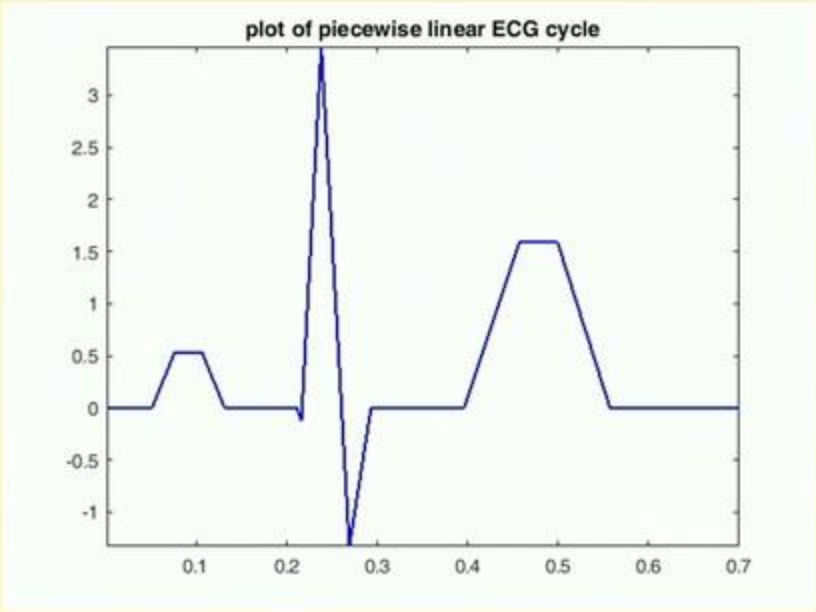
```
%% Generating piecewise linear ECG cycle
ecg_sel = ecg(1:700);                    % crop 1st cycle of ECG
x=[0,50,75,105,130,210,215,237,268,292,395,456,498,556,700];%Sample
                                                          %number
y = [0, 0, 0.201075269, 0.201075269, 0, 0, -0.046236559,
1.305017921, -0.501433692, 0, 0, 0.598924731, 0.598924731, 0, 0];  %
Amplitude
y = max(ecg_sel)*y;      % scale amplitude to make it same as input
                        %ECG signal
```

Solution 2.3

```
%% Generating piecewise linear ECG cycle
linModel =[];
for i = 1:length(x)-1        % for number of piecewise segments
 if isequal(y(i+1), y(i))                    % check for zero slope
    a = y(i)*ones(1,x(i+1) - x(i)); % for replicate previous values
 else
   a=y(i):(y(i+1)-y(i))/(x(i+1)-x(i)):y(i+1); % for non-zero slope
   a = a(1:end-1);                  % discarding last redundant point
 end
 linModel = [linModel,a];
End
t1 = (1:length(linModel))/fs; % time for plotting linear model
figure;
plot(t1,linModel); % plot piecewise linear ECG
```

Solution 2.3

- Generated piecewise linear ECG signal (desired signal)



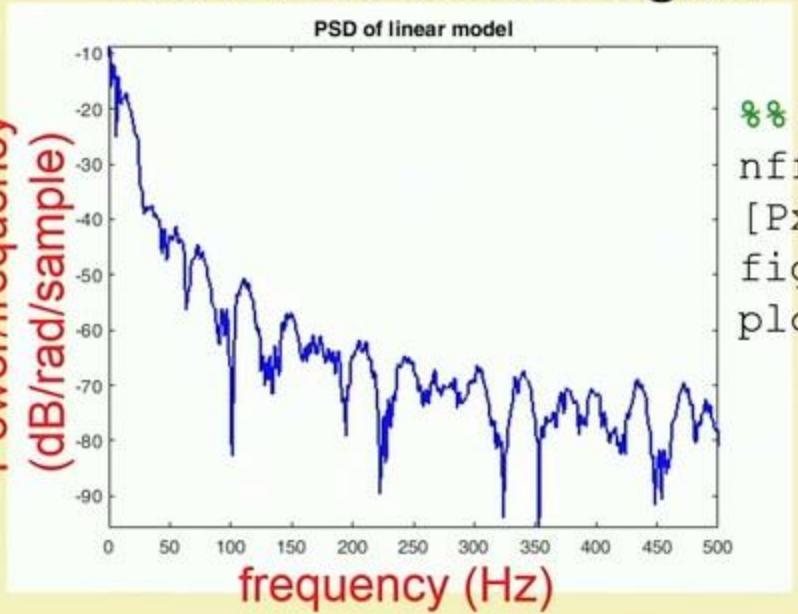
Solution 2.3

- The PSD of desired signal

```
%% PSD of desired signal
nfft=max(256,2^nextpow2(length(linModel)));
[Pxx,F] = periodogram(linModel,[],nfft,fs);
figure;
plot(F,10*log10(Pxx));
```

Solution 2.3

- The PSD of desired signal




```
%% PSD of desired signal
nfft=max(256,2^nextpow2(length(linModel)));
[Pxx,F] = periodogram(linModel,[],nfft,fs);
figure;
plot(F,10*log10(Pxx));
```

Solution 2.3

- The PSD of noise

```
%% PSD of desired signal
ECG = ecg(2776:2948);%TP segment of 1st cycle
ECG = ECG - mean(ECG);
[Pxx1,F]=periodogram(ECG,[],nfft,fs);
%% Average PSD of noise
ECG2 = ecg(4205:4381);%TP segment of 2nd
                      %cycle
ECG2 = ECG2 - mean(ECG2);
[Pxx2,F]=periodogram(ECG2,[],nfft,fs);
ECG3 = ecg(7051:7230);%TP segment of 3rd
                      %cycle
ECG3 = ECG3 - mean(ECG3);
[Pxx3,F]=periodogram(ECG3,[],nfft,fs);
Pxx_avg = (Pxx1+Pxx2+Pxx3)/3;% Averaging PSD
                             of noise
figure;  plot(F,10*log10(Pxx_avg));
```

Solution 2.3

- The PSD of noise

```
%% PSD of desired signal
ECG = ecg(2776:2948);%TP segment of 1st cycle
ECG = ECG - mean(ECG);
[Pxx1,F]=periodogram(ECG,[],nfft,fs);
%% Average PSD of noise
ECG2 = ecg(4205:4381);%TP segment of 2nd
                      %cycle
ECG2 = ECG2 - mean(ECG2);
[Pxx2,F]=periodogram(ECG2,[],nfft,fs);
ECG3 = ecg(7051:7230);%TP segment of 3rd
                      %cycle
ECG3 = ECG3 - mean(ECG3);
[Pxx3,F]=periodogram(ECG3,[],nfft,fs);
Pxx_avg = (Pxx1+Pxx2+Pxx3)/3;% Averaging PSD
                             of noise
figure;  plot(F,10*log10(Pxx_avg));
```

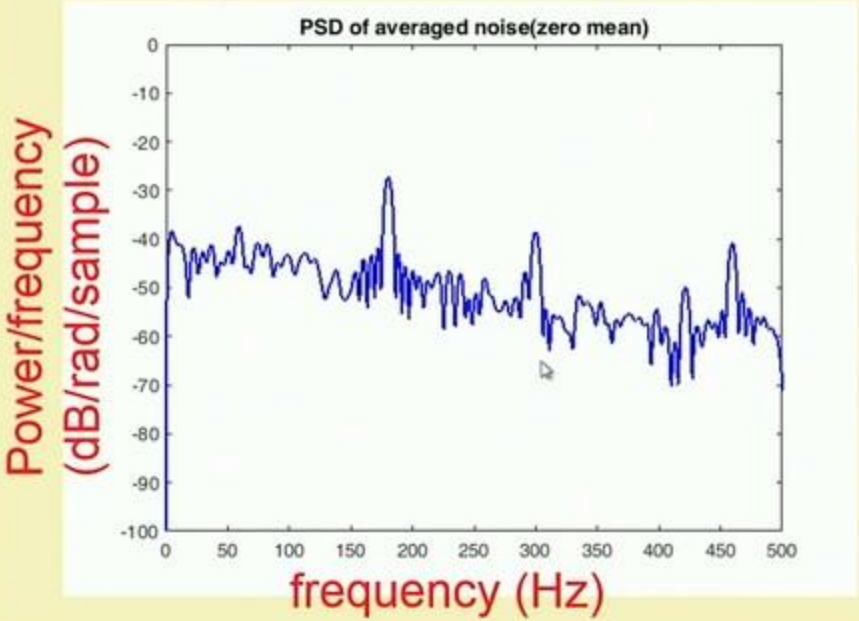
Solution 2.3

- The PSD of noise

```
%% PSD of desired signal
ECG = ecg(2776:2948);%TP segment of 1st cycle
ECG = ECG - mean(ECG);
[Pxx1,F]=periodogram(ECG,[],nfft,fs);
%% Average PSD of noise
ECG2 = ecg(4205:4381);%TP segment of 2nd
                      %cycle
ECG2 = ECG2 - mean(ECG2);
[Pxx2,F]=periodogram(ECG2,[],nfft,fs);
ECG3 = ecg(7051:7230);%TP segment of 3rd
                      %cycle
ECG3 = ECG3 - mean(ECG3);
[Pxx3,F]=periodogram(ECG3,[],nfft,fs);
Pxx_avg = (Pxx1+Pxx2+Pxx3)/3;% Averaging PSD
                             of noise
figure;  plot(F,10*log10(Pxx_avg));
```

Solution 2.3

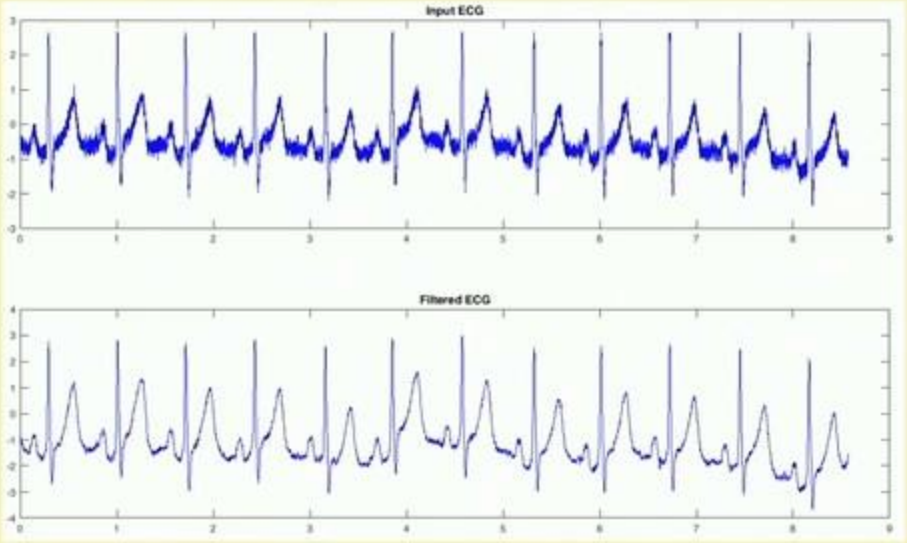
- The PSD of noise




```
%% PSD of desired signal
ECG = ecg(2776:2948);%TP segment of 1st cycle
ECG = ECG - mean(ECG);
[Pxx1,F]=periodogram(ECG,[],nfft,fs);
%% Average PSD of noise
ECG2 = ecg(4205:4381);%TP segment of 2nd
                      %cycle
ECG2 = ECG2 - mean(ECG2);
[Pxx2,F]=periodogram(ECG2,[],nfft,fs);
ECG3 = ecg(7051:7230);%TP segment of 3rd
                      %cycle
ECG3 = ECG3 - mean(ECG3);
[Pxx3,F]=periodogram(ECG3,[],nfft,fs);
Pxx_avg = (Pxx1+Pxx2+Pxx3)/3;% Averaging PSD
                             of noise
figure;  plot(F,10*log10(Pxx_avg));
```

Solution 2.3

- Wiener filtering

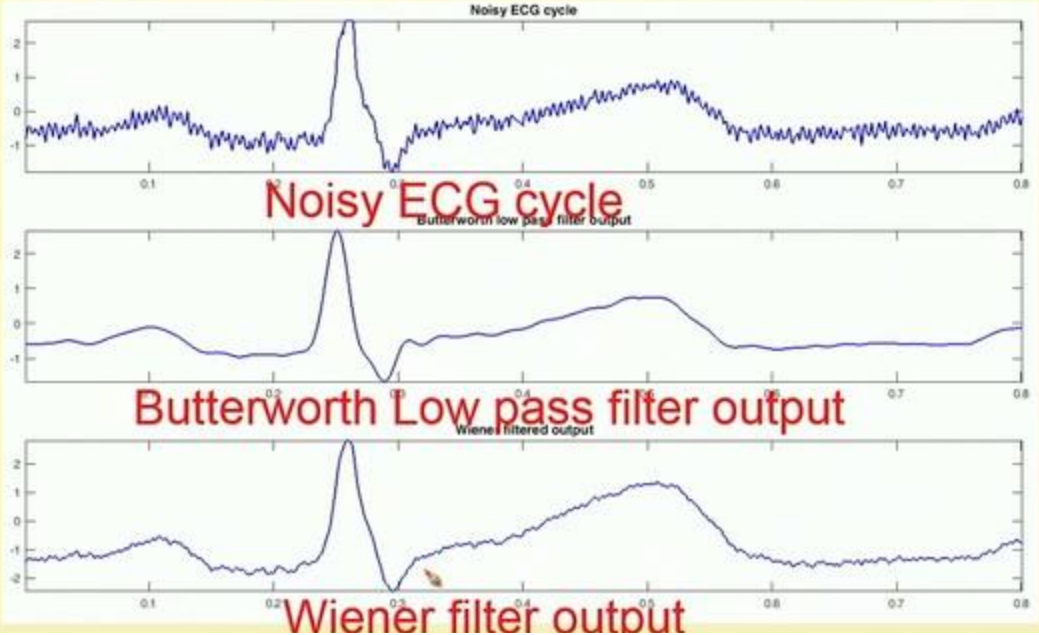



```
%% transfer function of Wiener filter
W = zeros(1,length(F));
for i = 1:length(F)
    W(i) = 1/(1+(Pxx_avg(i)/Pxx(i)));
end
%W in time domain
Y = ifftshift(abs(ifft(W,200)));

output = conv(ecg,Y);
```

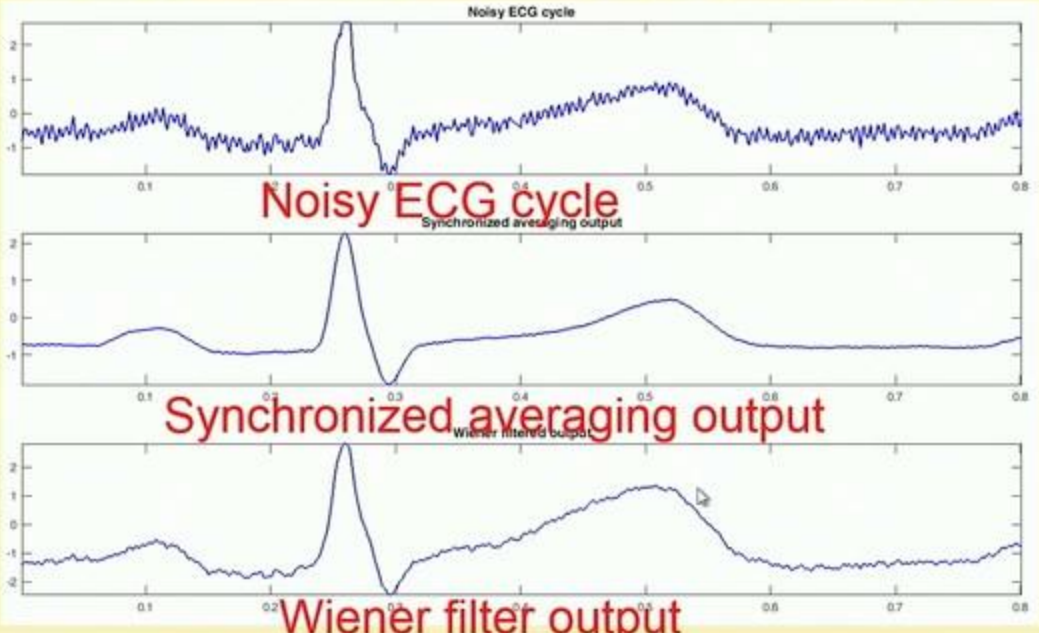
Solution 2.3

- Comparison of Butterworth low pass filter output with Wiener filter output



Solution 2.3

- Comparison of Synchronized averaging output with Wiener filter output



Solution 2.3

Observations

- The wiener filter is able to suppress the noise but not able to remove completely
- The output of synchronized averaging is more smoother than wiener filter
- The output of low pass filter is smoother but slightly distorted compared to wiener filter

Question 3.1

- Implement the Pan-Tompkins method for QRS detection in MATLAB, you may employ a simple threshold-based method to detect QRS complexes as the procedure will be run off-line. Apply the procedure to the signals in the files ECG3.dat, ECG4.dat, ECG5.dat, and ECG6.dat, sampled at a rate of 200 Hz. Compute the averaged heart rate and QRS width for each record. Verify your results by measuring the parameters visually from plots of the signals.

Solution 3.1

- Four input ECG signals (ecg3.dat, ecg4.dat, ecg5.dat, and ecg6.dat) are available at:
  http://people.ucalgary.ca/~ranga/enel563/SIGNAL_DATA_FILES/

- Sample MATLAB code to read and display the input ECG is available at:
  http://people.ucalgary.ca/~ranga/enel563/SIGNAL_DATA_FILES/ECGS.m

Note: Keep the input signal and the MATLAB codes in the same directory.

Solution 3.1

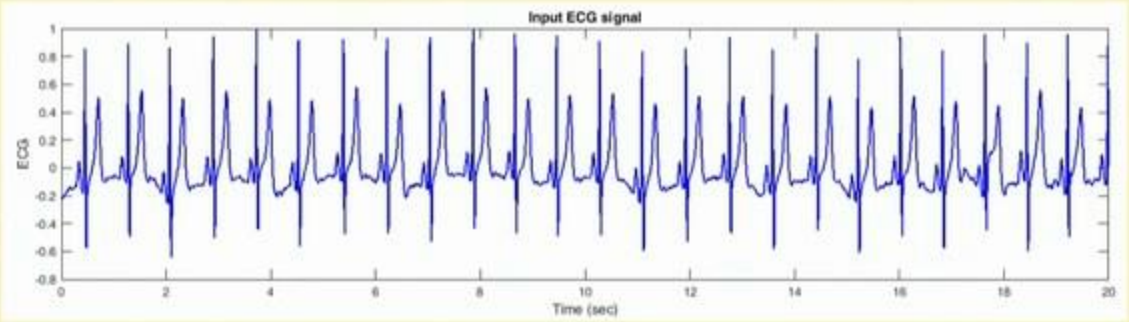
Read Input ECG signal

- Read ECG data

```
ecg = load('ecg3.dat');
ecg = ecg - mean(ecg);
ecg = ecg/max(abs(ecg));
fs = 200;%sampling rate
slen = length(ecg);
t=[1:slen]/fs; %time
plot(t,ecg)
```

Solution 3.1

Read Input ECG signal

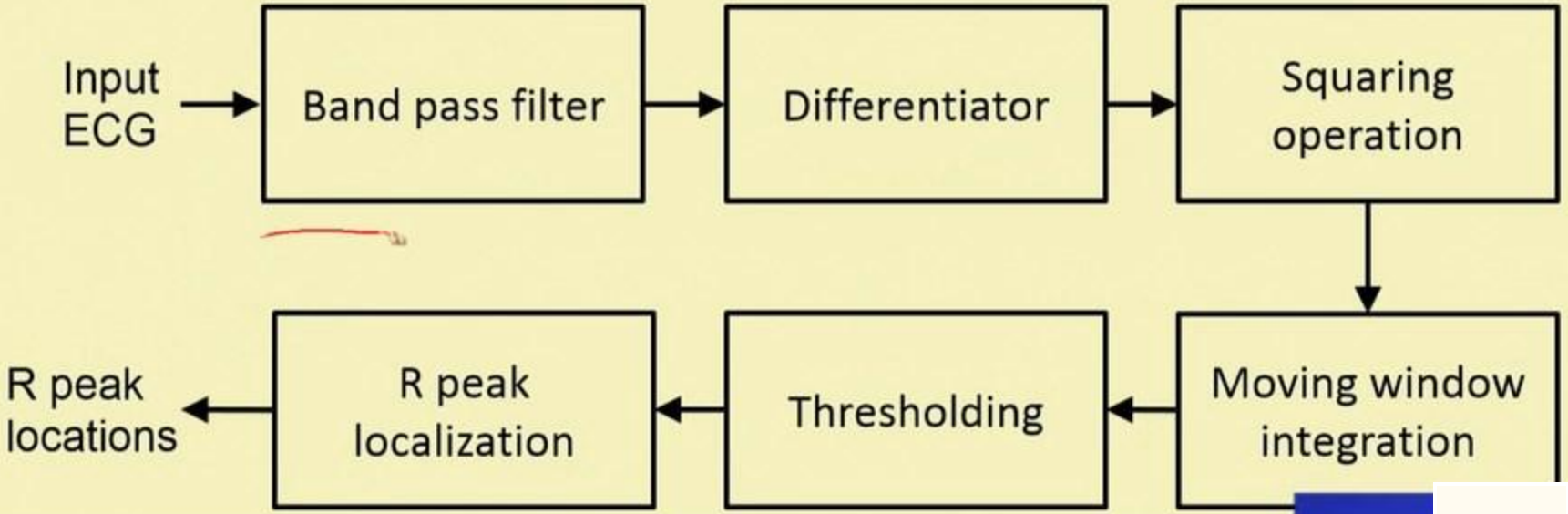


- Read ECG data

```
ecg = load('ecg3.dat');
ecg = ecg - mean(ecg);
ecg = ecg/max(abs(ecg));
fs = 200;%sampling rate
slen = length(ecg);
t=[1:slen]/fs; %time
plot(t,ecg)
```

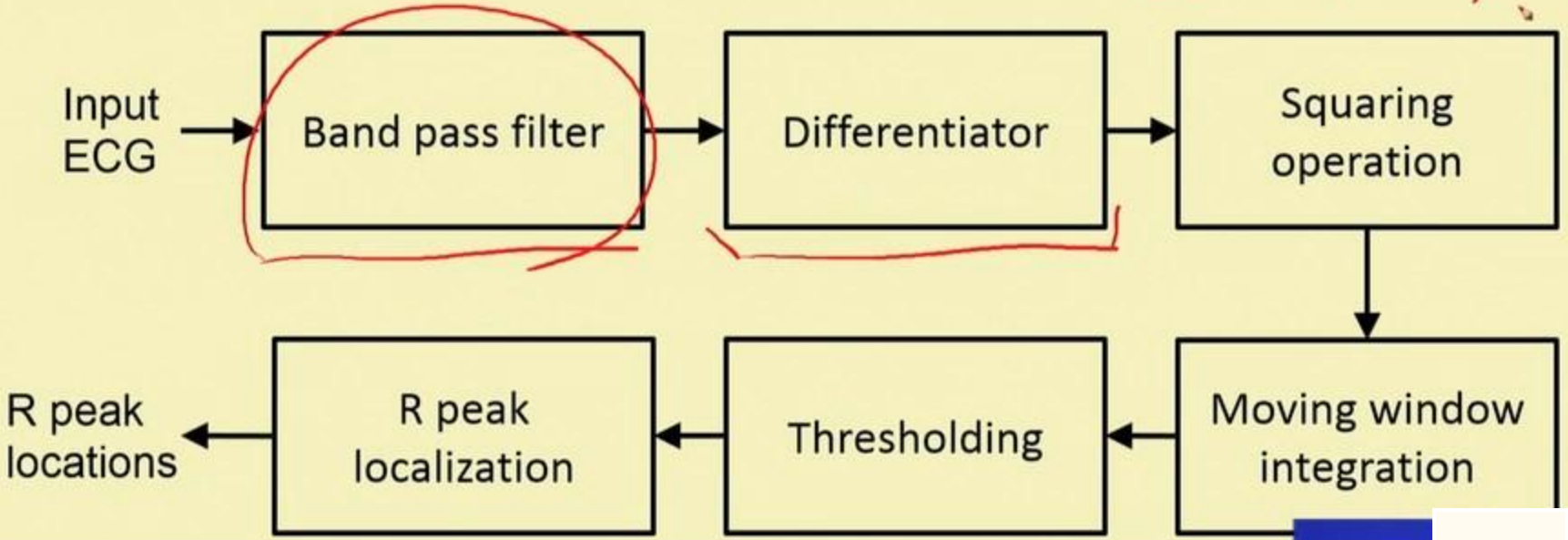
Solution 3.1

- Block diagram of Pan Tompkins algorithm



Solution 3.1

- Block diagram of Pan Tompkins algorithm



Solution 3.1

1. Band pass filtering

```
%% Low Pass Filter %%
b=[1 0 0 0 0 0 -2 0 0 0 0 0 1];
a=[1 -2 1] * 32;
%Creating digital filter(Direct Form II)
hd1=dfilt.df2(b,a);
%Filtering the ECG signal using LPF
ecg_out1=filter(hd1,ecg);
ecg_out1=ecg_out1-mean(ecg_out1);
ecg_out1=ecg_out1/max(abs(ecg_out1));
```

Solution 3.1

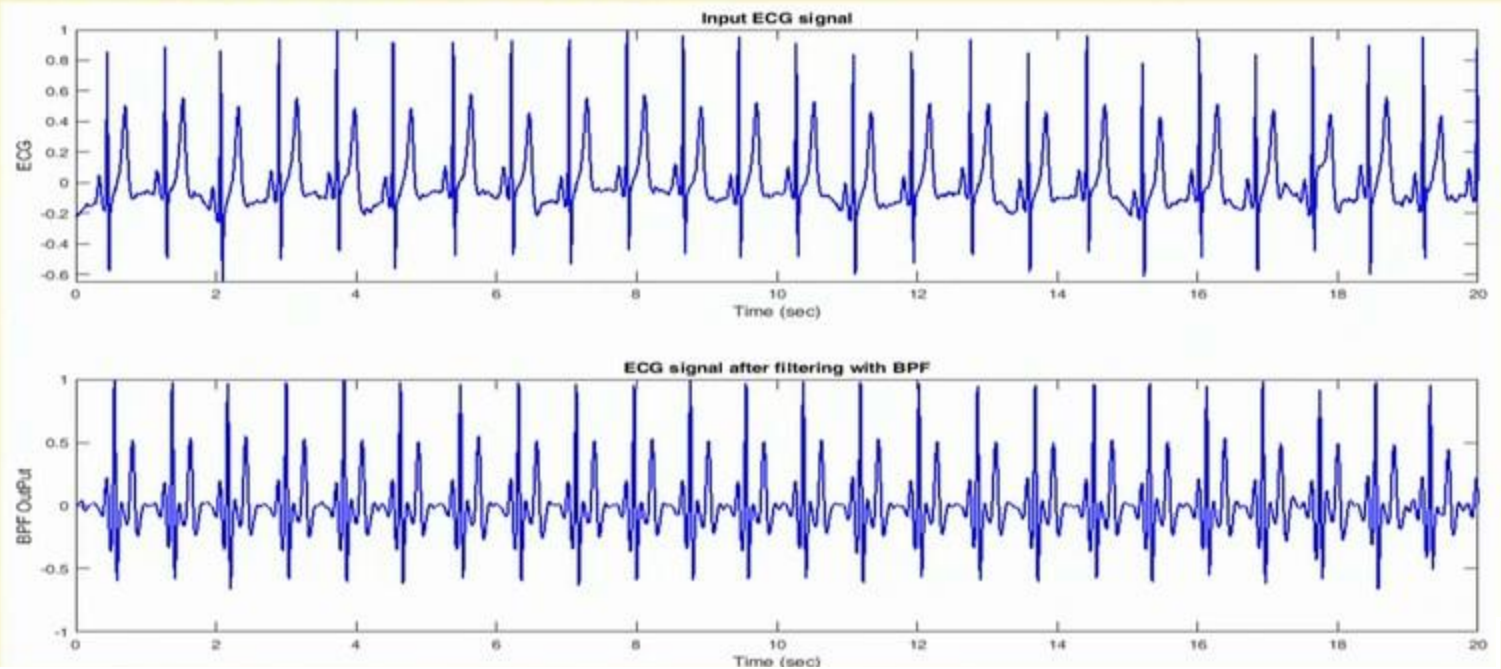
1. Band pass filtering

```
%% Low Pass Filter %%
b=[1 0 0 0 0 0 -2 0 0 0 0 0 1];
a=[1 -2 1] * 32;
%Creating digital filter(Direct Form II)
hd1=dfilt.df2(b,a);
%Filtering the ECG signal using LPF
ecg_out1=filter(hd1,ecg);
ecg_out1=ecg_out1-mean(ecg_out1);
ecg_out1=ecg_out1/max(abs(ecg_out1));
```

```
%% High Pass Filter %%
b2=[-1,zeros(1,15),32,-32,zeros(1,14),1];
a2=[1 -1] * 32;
%Creating digital filter (Direct Form II)
hd2 = dfilt.df2(b2,a2);
%Filtering the ECG signal from HPF
ecg_out2 = filter(hd2,ecg_out1);
ecg_out2 = [ecg_out2(1:40)*0.25;
ecg_out2(41:end)];
ecg_out2 = ecg_out2/max(abs(ecg_out2));
```

Solution 3.1

1. Band pass filtering output



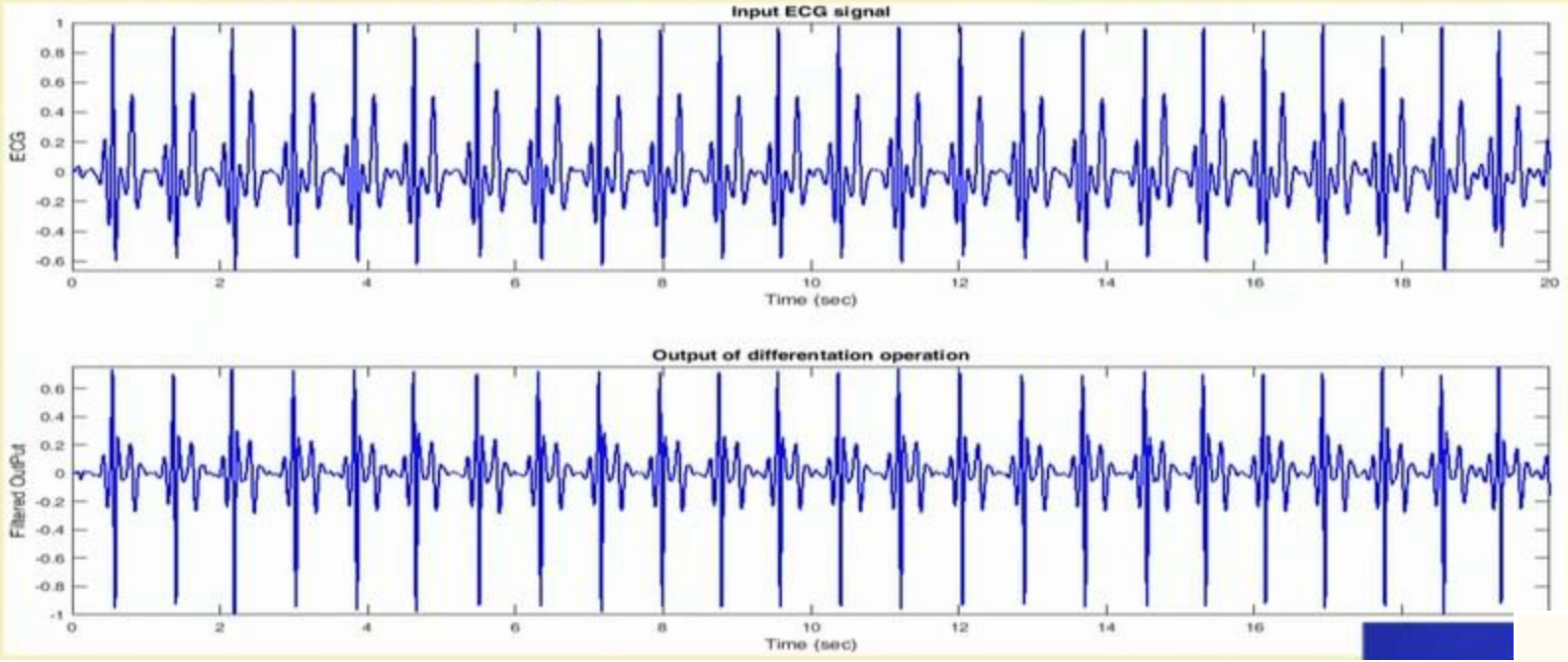
Solution 3.1

2. Differentiator

```
%% Derivative Operator
b3 = [2 1 0 -1 -2];
a3 = [1 ] * 8;
% Creating digital filter (Direct Form II)
hd3 = dfilt.df2(b3,a3);
% Filtering the ECG signal from derivative operator
ecg_out3 = filter(hd3,ecg_out2);
ecg_out3 = ecg_out3/max(abs(ecg_out3));
```

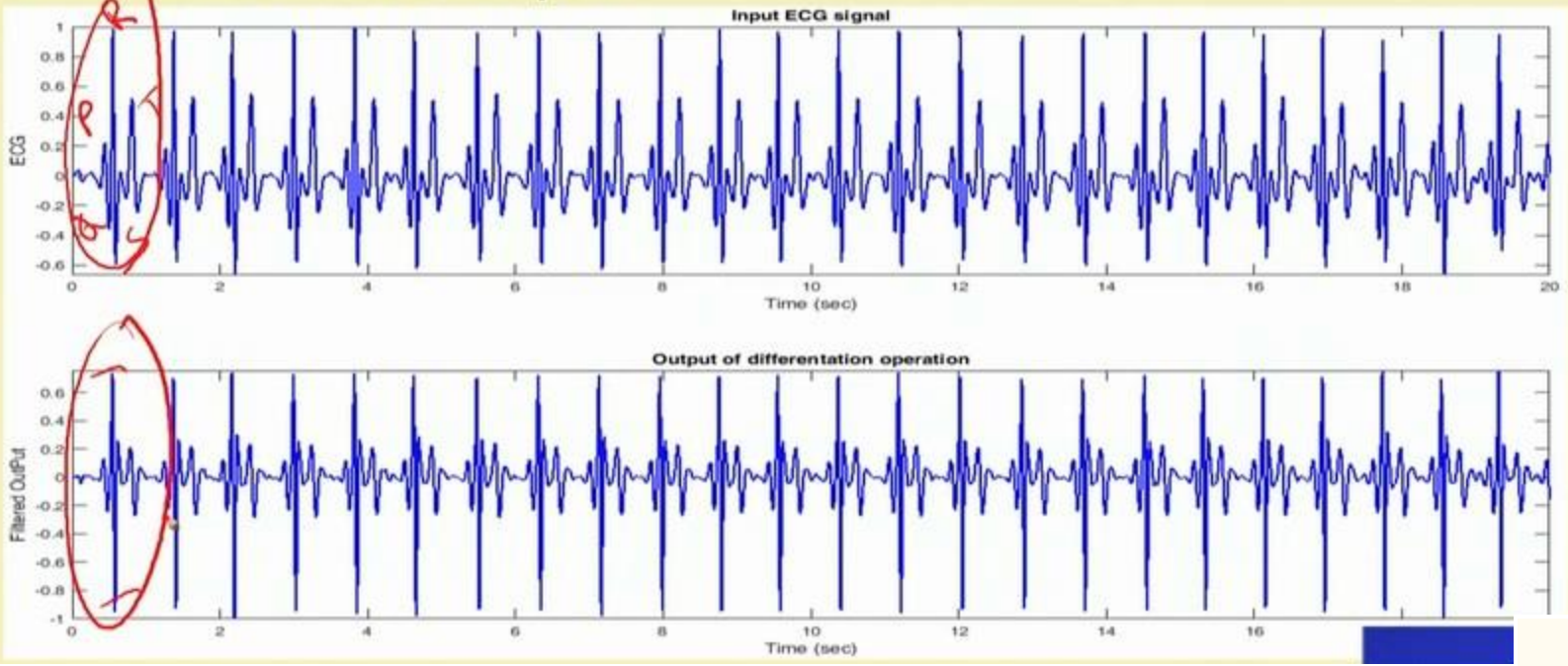
Solution 3.1

2. Differentiator output



Solution 3.1

2. Differentiator output



Solution 3.1

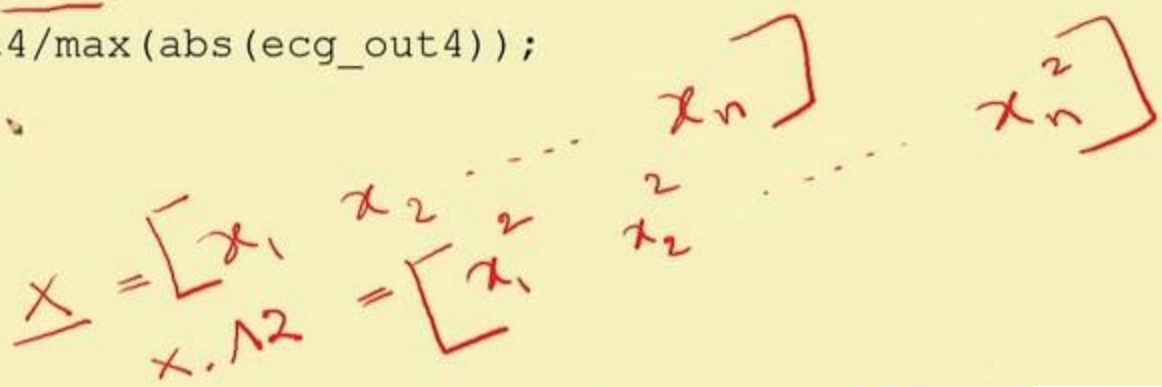
3. Squaring operation

```
%% Squaring
ecg_out4 = ecg_out3.^2;
ecg_out4 = ecg_out4/max(abs(ecg_out4));
```

Solution 3.1

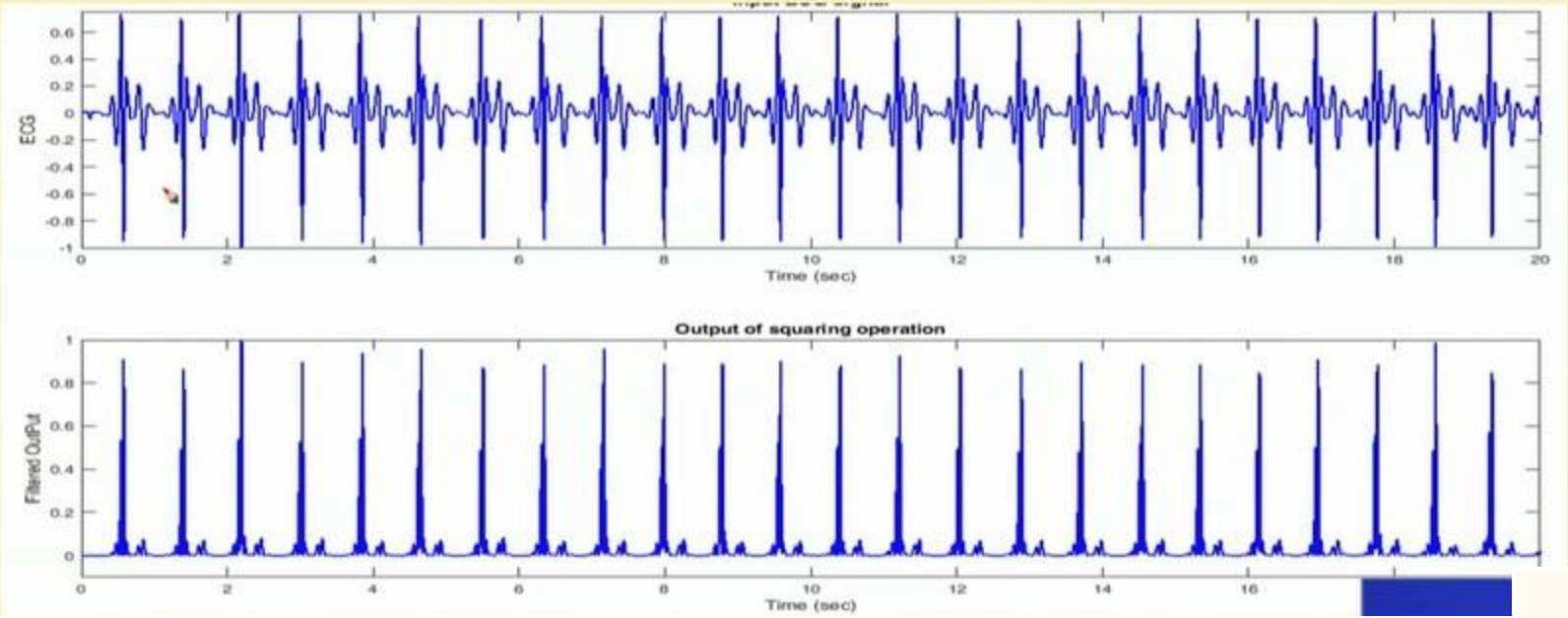
3. Squaring operation

```
%% Squaring
ecg_out4 = ecg_out3.^2;
ecg_out4 = ecg_out4/max(abs(ecg_out4));
```



Solution 3.1

3. Squaring operation



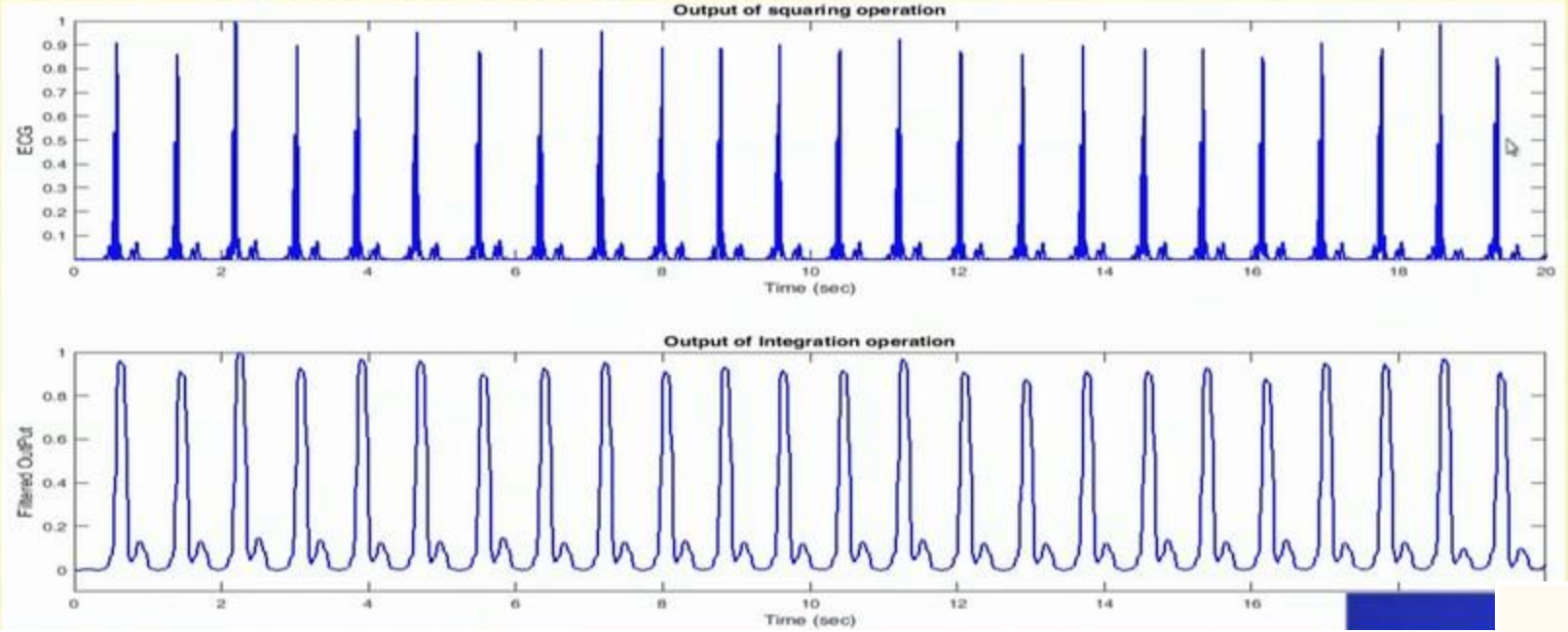
Solution 3.1

4. Moving window integration

```
%% Moving window integration Operator
ecg_out4pad = [zeros(1,29) ecg_out4' zeros(1,29)];
for i=30:length(ecg_out4pad)-29
    ecg5(i-29) = sum(ecg_out4pad(i-29:i))/30;
end
ecg5 = ecg5';
ecg5 = ecg5/max(abs(ecg5));
```

Solution 3.1

4. Moving window integration



Solution 3.1

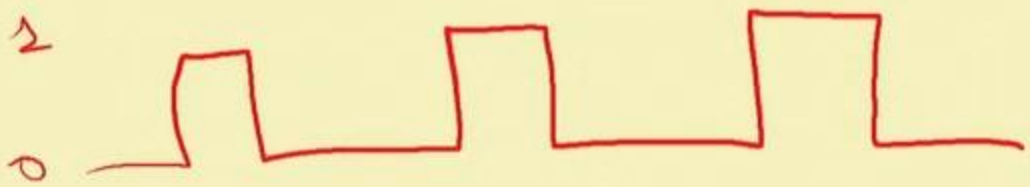
5. Thresholding

```
TH = mean(ecg5);
ecg6 = zeros(slen,1);
w = (ecg5>(TH));
ecg6(w) = 1;
x = find(diff([0 w']) == 1);
y = find(diff([w' 0]) == -1);
% cancel delay because of LP and HP
x = x - (6+16);
y = y - (6+16);
```

Solution 3.1

5. Thresholding

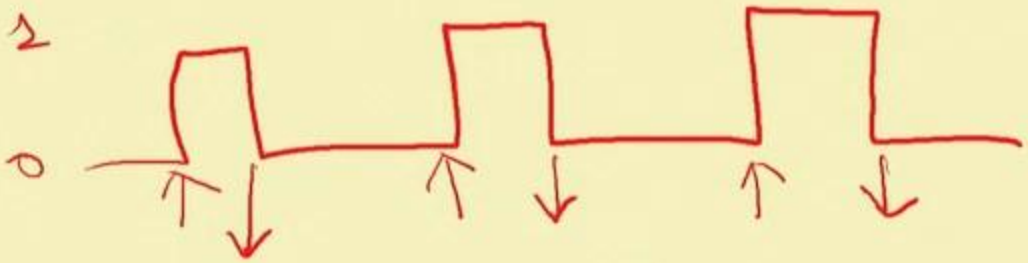
```
TH = mean(ecg5);
ecg6 = zeros(slen,1);
w = (ecg5>(TH));
ecg6(w) = 1;
x = find(diff([0 w']) == 1);
y = find(diff([w' 0]) == -1);
% cancel delay because of LP and HP
x = x - (6+16);
y = y - (6+16);
```



Solution 3.1

5. Thresholding

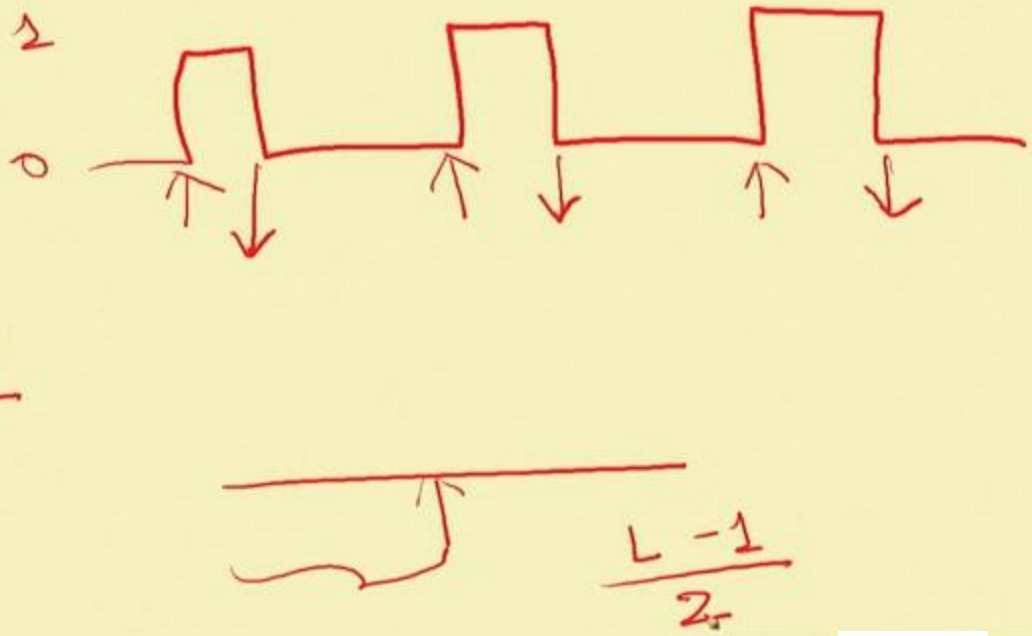
```
TH = mean(ecg5);
ecg6 = zeros(slen,1);
w = (ecg5>(TH));
ecg6(w) = 1;
x = find(diff([0 w']) == 1);
y = find(diff([w' 0]) == -1);
% cancel delay because of LP and HP
x = x - (6+16);
y = y - (6+16);
```



Solution 3.1

5. Thresholding

```
TH = mean(ecg5);
ecg6 = zeros(slen,1);
w = (ecg5>(TH));
ecg6(w) = 1;
x = find(diff([0 w']) == 1);
y = find(diff([w' 0]) == -1);
% cancel delay because of LP and HP
x = x - (6+16);
y = y - (6+16);
```



Solution 3.1

5. Thresholding

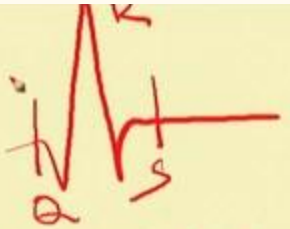
```
TH = mean(ecg5);
ecg6 = zeros(slen,1);
w = (ecg5>(TH));
ecg6(w) = 1;
x = find(diff([0 w']) == 1);
y = find(diff([w' 0]) == -1);
% cancel delay because of LP and HP
x = x - (6+16);
y = y - (6+16);
```

```
% Detect R,Q, and S points
for i=1:length(x)
[R_val(i),R_loc(i)]=max(ecg(x(i):y(i)));
 R_loc(i)=R_loc(i)-1+x(i); % add offset
[Q_val(i),Q_loc(i)]=min(ecg(R_loc(i):-
1:R_loc(i)-8));
Q_loc(i) = R_loc(i) - Q_loc(i)+1;
[S_val(i),S_loc(i)]=min(ecg(R_loc(i):
R_loc(i)+10));
S_loc(i) = R_loc(i) + S_loc(i)-1;
end
```

Solution 3.1

5. Thresholding

```
TH = mean(ecg5);
ecg6 = zeros(slen,1);
w = (ecg5>(TH));
ecg6(w) = 1;
x = find(diff([0 w']) == 1);
y = find(diff([w' 0]) == -1);
% cancel delay because of LP and HP
x = x - (6+16);
y = y - (6+16);
```

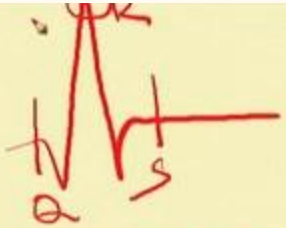



```
% Detect R,Q, and S points
for i=1:length(x)
[R_val(i),R_loc(i)]=max(ecg(x(i):y(i)));
 R_loc(i)=R_loc(i)-1+x(i); % add offset
[Q_val(i),Q_loc(i)]=min(ecg(R_loc(i):-
1:R_loc(i)-8));
Q_loc(i) = R_loc(i) - Q_loc(i)+1;
[S_val(i),S_loc(i)]=min(ecg(R_loc(i):
R_loc(i)+10));
S_loc(i) = R_loc(i) + S_loc(i)-1;
end
```

Solution 3.1

5. Thresholding

```
TH = mean(ecg5);
ecg6 = zeros(slen,1);
w = (ecg5>(TH));
ecg6(w) = 1;
x = find(diff([0 w']) == 1);
y = find(diff([w' 0]) == -1);
% cancel delay because of LP and HP
x = x - (6+16);
y = y - (6+16);
```




```
% Detect R,Q, and S points
for i=1:length(x)
[R_val(i),R_loc(i)]=max(ecg(x(i):y(i)));
 R_loc(i)=R_loc(i)-1+x(i); % add offset
[Q_val(i),Q_loc(i)]=min(ecg(R_loc(i):-
1:R_loc(i)-8));
Q_loc(i) = R_loc(i) - Q_loc(i)+1;
[S_val(i),S_loc(i)]=min(ecg(R_loc(i):
R_loc(i)+10));
S_loc(i) = R_loc(i) + S_loc(i)-1;
end
```

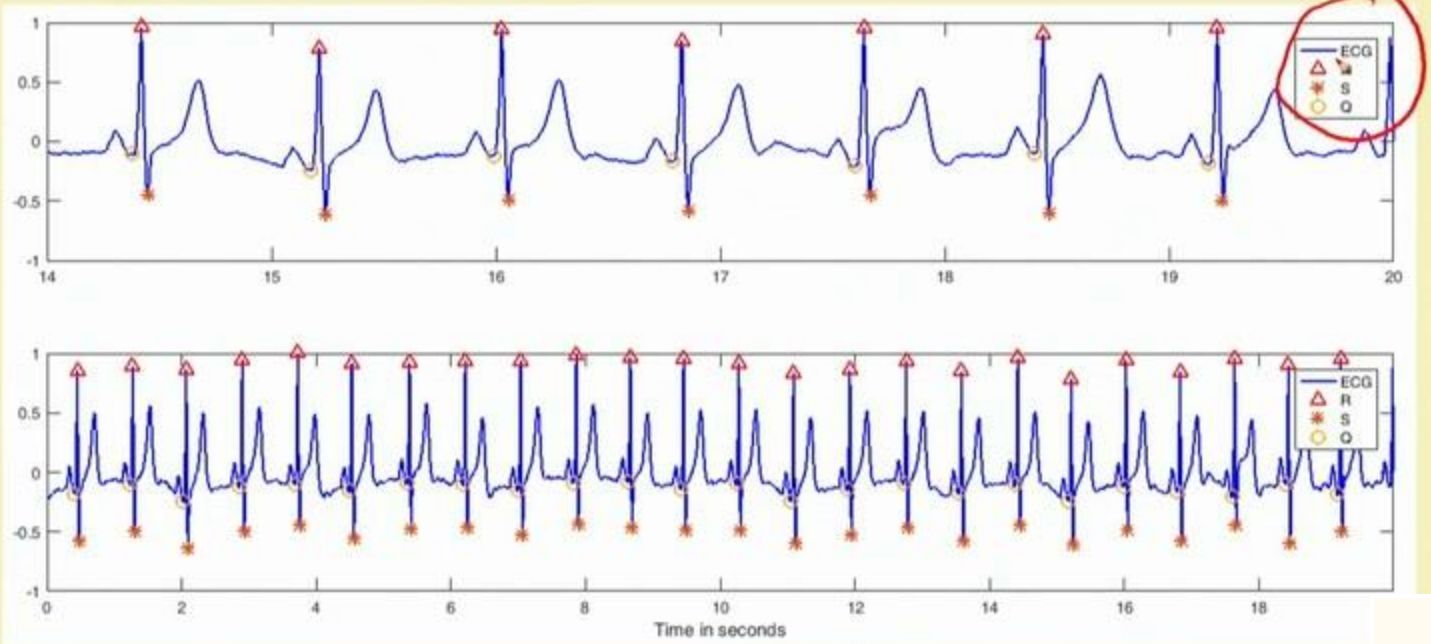
Solution 3.1

6. R peak localization

```
% Plotting ECG signal with Q,R, and S points marked
subplot(2,1,1)
plot(t,ecg/max(ecg),t(R_loc),R_val,'r^',t(S_loc),S_val,'*',t(Q_loc),Q_val,'o');
xlim([14 20]); legend('ECG','R','S','Q');
subplot(2,1,2);
%title('ECG Signal with R points');
plot(t,ecg/max(ecg),t(R_loc),R_val,'r^',t(S_loc),S_val,'*',t(Q_loc),Q_val,'o');
legend('ECG','R','S','Q');xlabel('Time in seconds');
```

Solution 3.1

6. R peak localization results



Solution 3.1

Calculation of QRS duration and heart rate

```
count = 0;
for i=1:slen-1
    if(ecg6(i) == 0 && ecg6(i+1)>= 1 && ecg6(i+2) >= 1)
        count = count + 1;
    end
end
ecg7 = diff([ecg6; 0]);
```

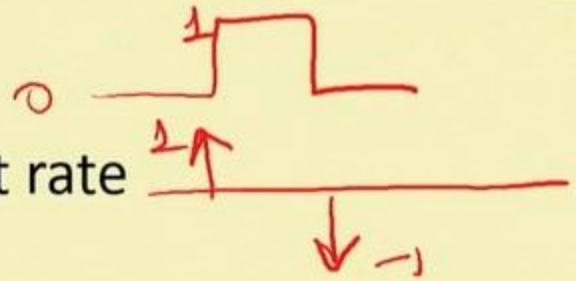
Solution 3.1

Calculation of QRS duration and heart rate

```
x = find(ecg7>0);
y = find(ecg7<0);
z = y - x;
dur = mean(z)*(1/fs);
disp(['QRS Duration for ecg3 = ' num2str(dur) ' sec']);
HT = count * 3;
disp(['Heart rate for ecg3 = ' num2str(HT) ' beats/min']);
```

Solution 3.1

Calculation of QRS duration and heart rate




```
x = find(ecg7>0);
y = find(ecg7<0);
z = y - x;
dur = mean(z)*(1/fs);
disp(['QRS Duration for ecg3 = ' num2str(dur) ' sec']);
HT = count * 3;
disp(['Heart rate for ecg3 = ' num2str(HT) ' beats/min']);
```

Solution 3.1

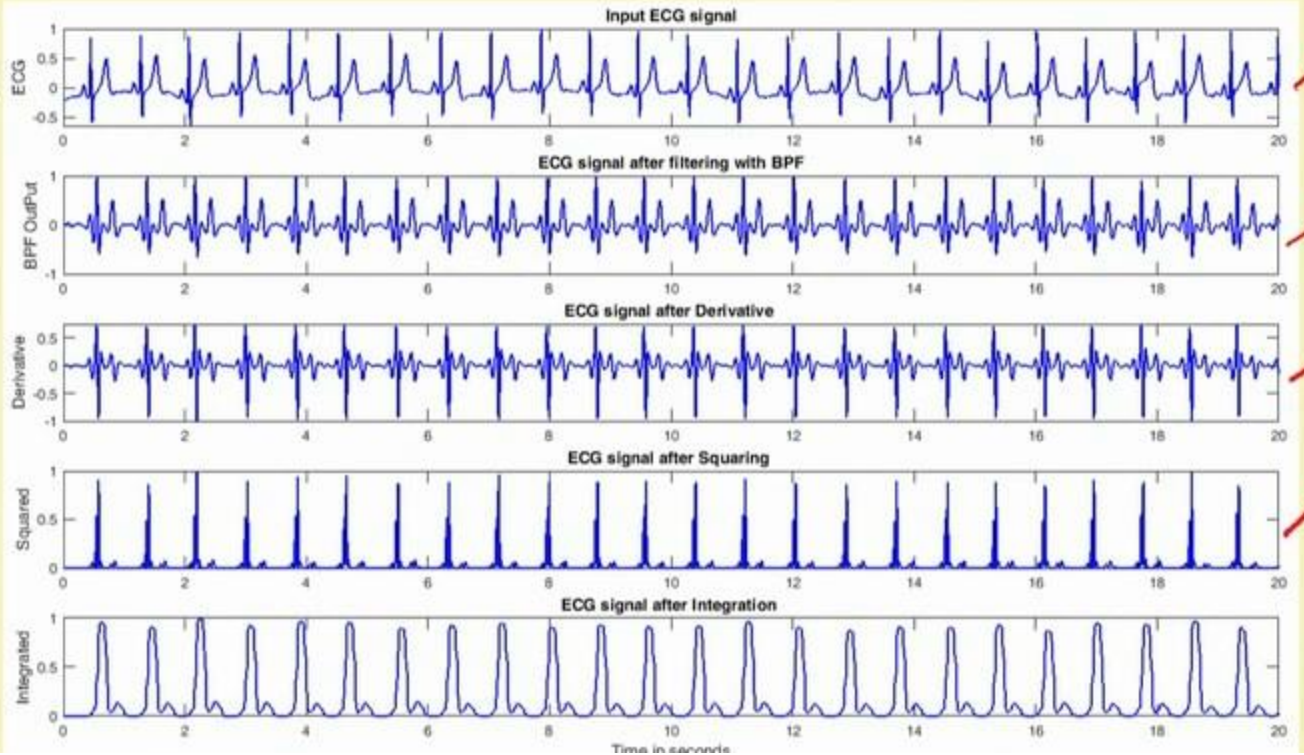
Results on command window

Command Window

```
QRS Duration for ecg3 = 0.19312 sec
heart rate for ecg3 = 72 beats/min
QRS Duration for ecg4 = 0.18758 sec
heart rate for ecg4 = 93 beats/min
QRS Duration for ecg5 = 0.16467 sec
heart rate for ecg5 = 135 beats/min
QRS Duration for ecg6 = 0.19917 sec
heart rate for ecg6 = 54 beats/min
>>
```

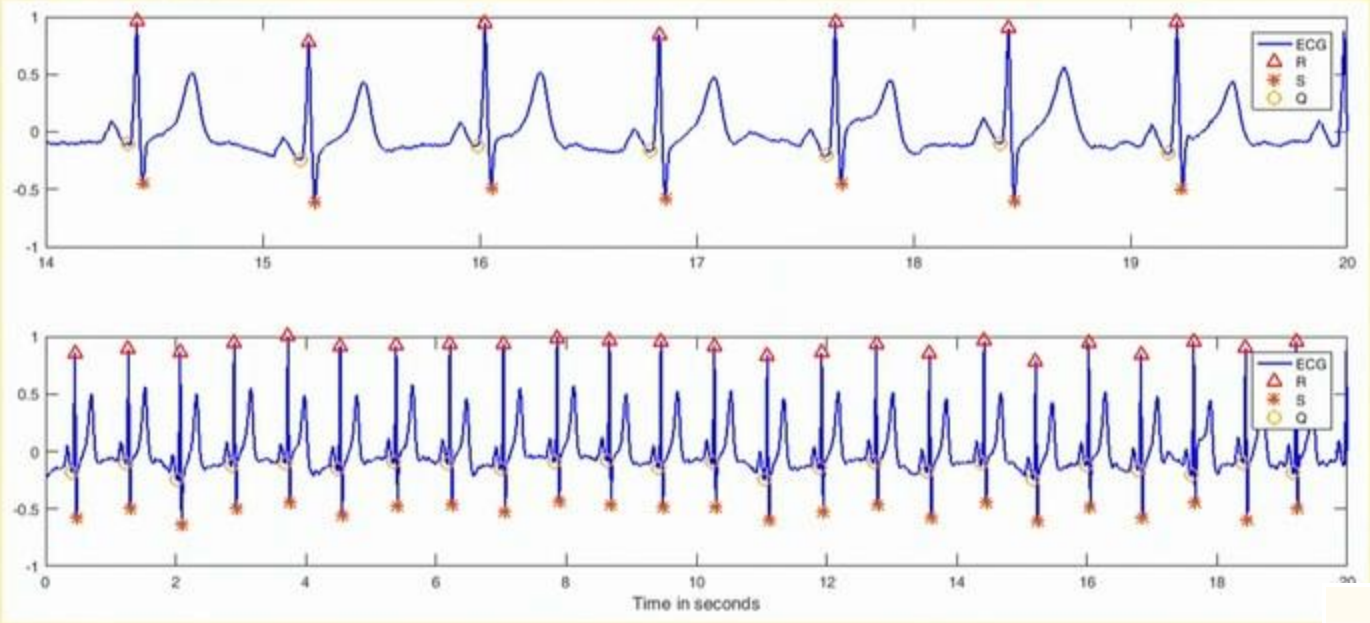
Solution 3.1

Results of ECG3



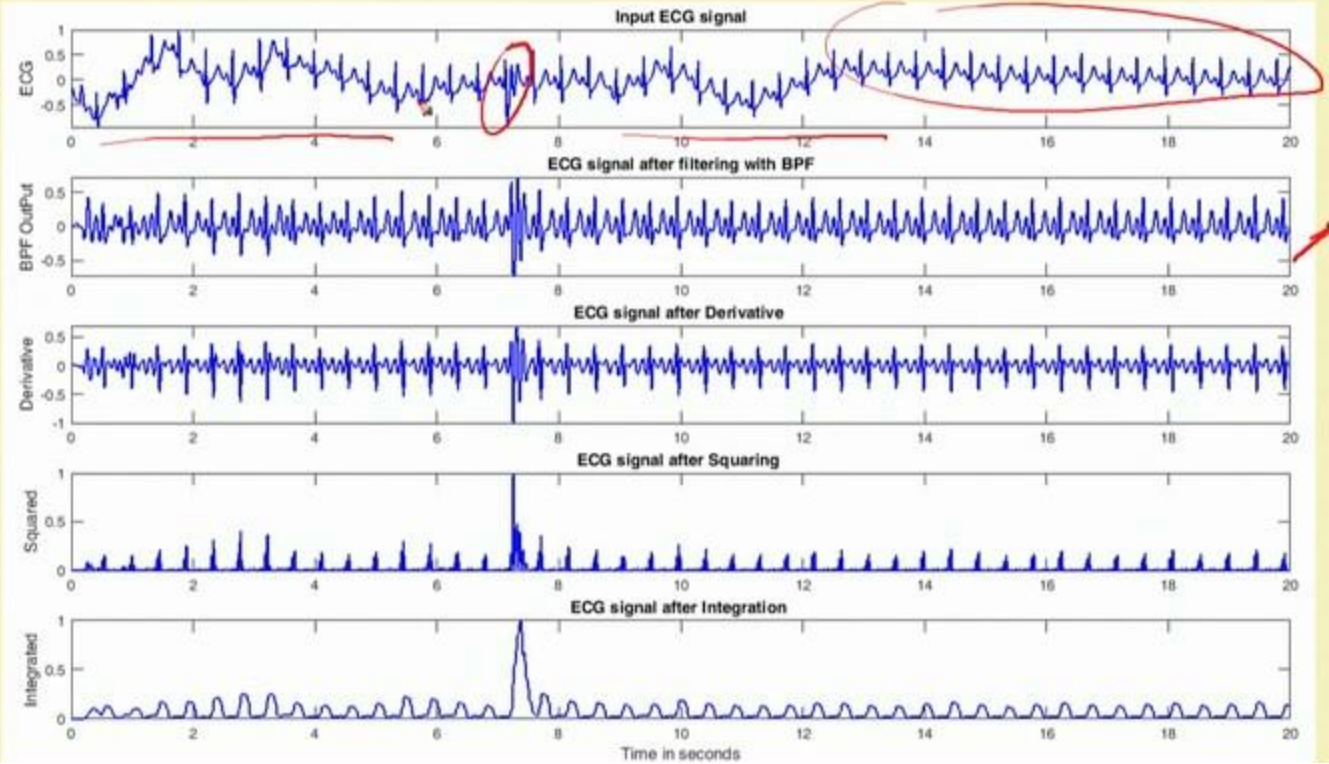
Solution 3.1

Results of ECG3



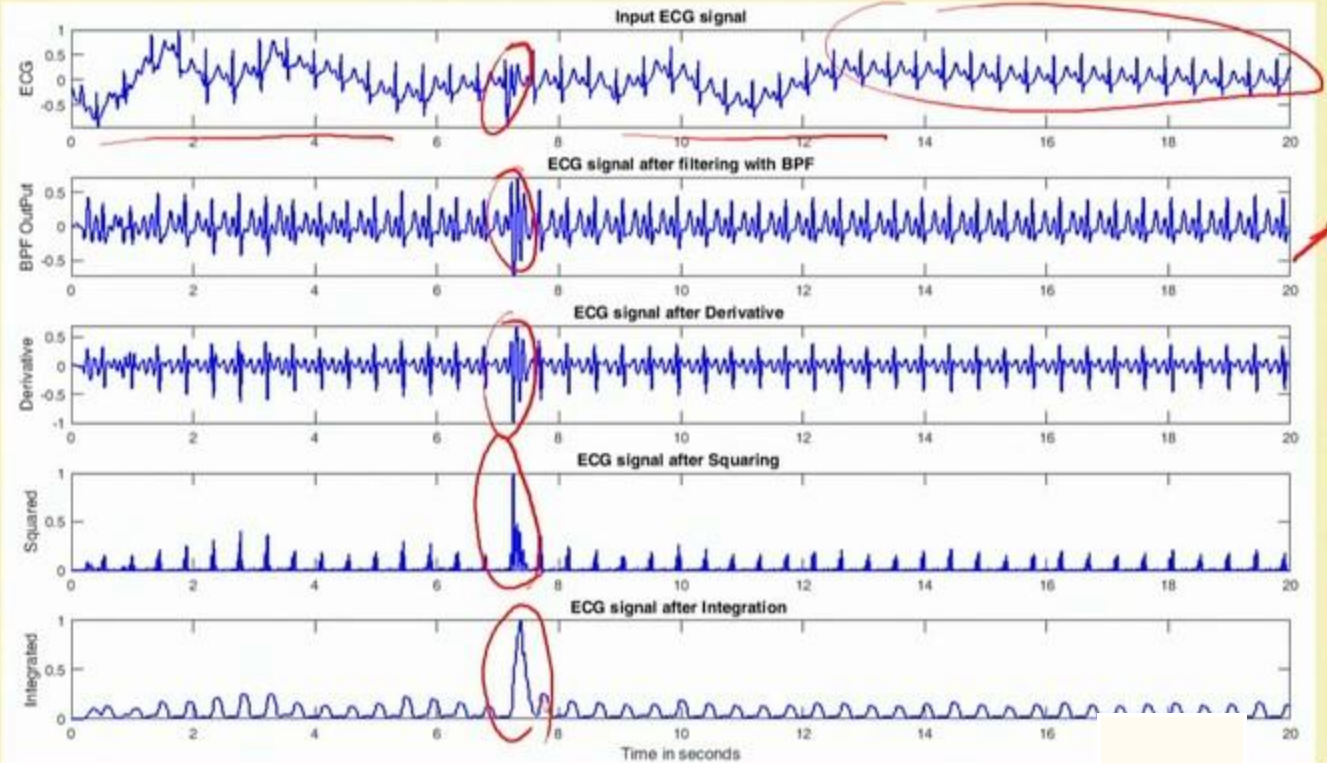
Solution 3.1

Results of ECG5



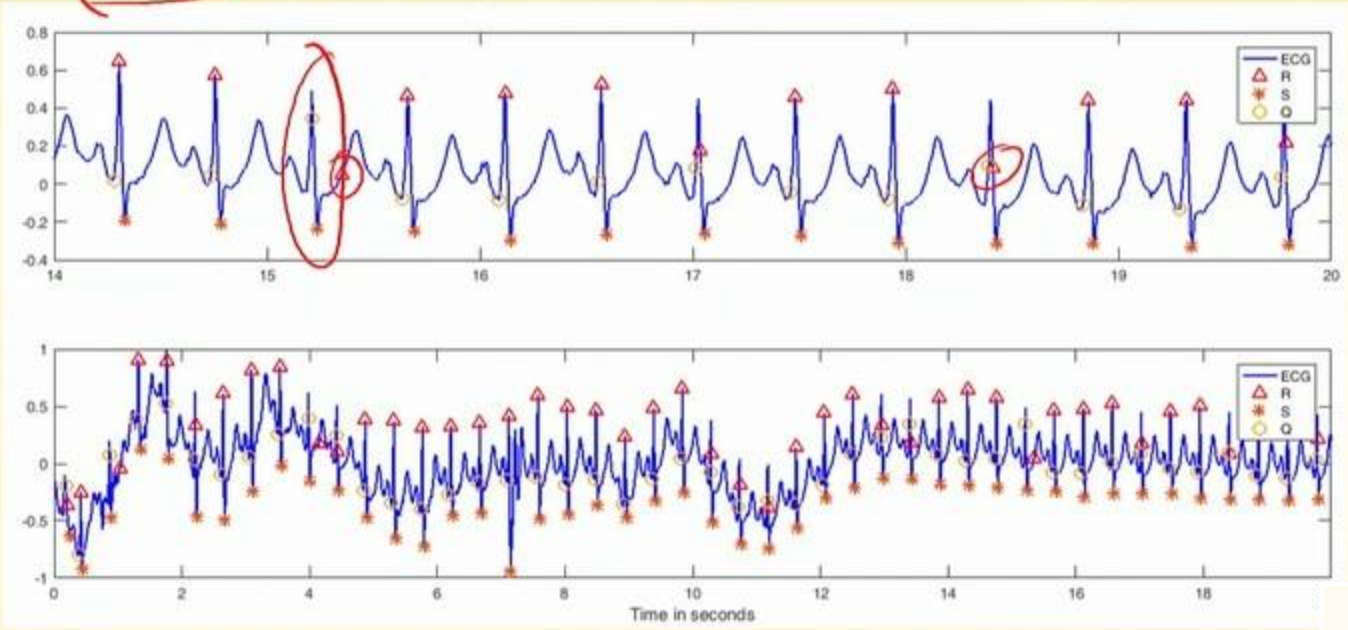
Solution 3.1

Results of ECG5



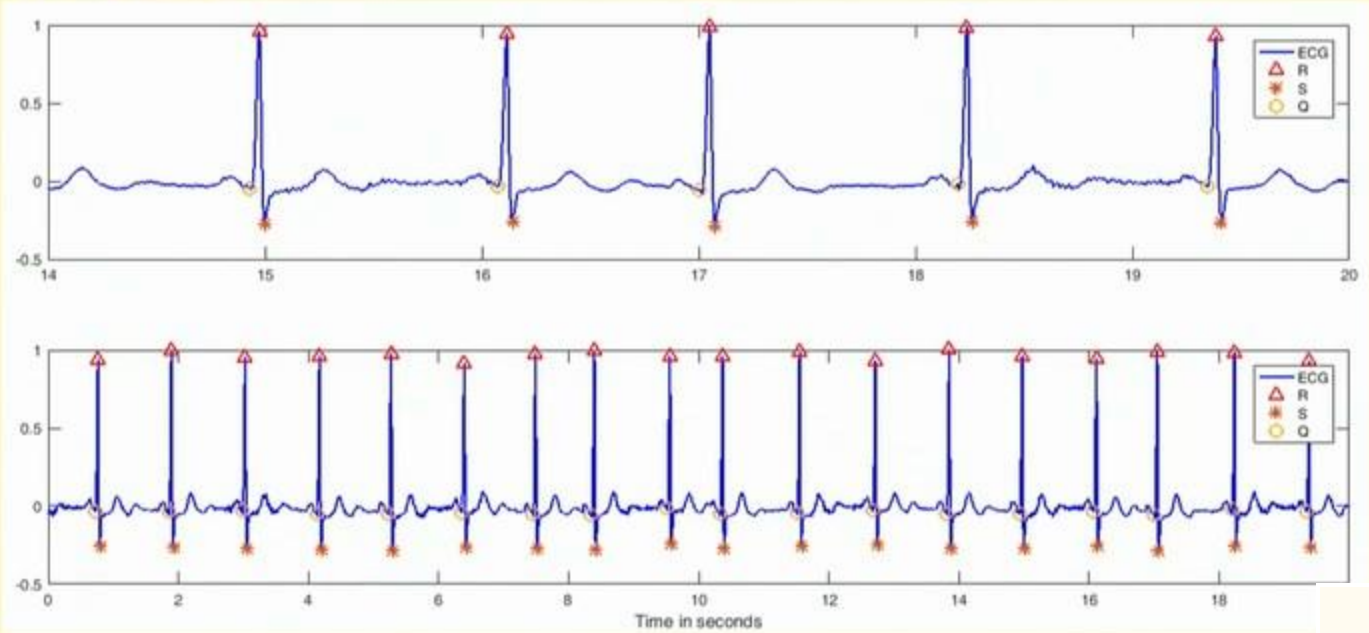
Solution 3.1

Results of ECG5



Solution 3.1

Results of ECG6



Solution 3.1

Observations

- Pan Tompkins algorithm is able to detect QRS wave of all ECG beats in 3 out of 4 signals
- Pan Tompkins algorithm is able to detect QRS wave even in presence of low frequency artifact and irregular occurrence of ECG beats in the signal
- Pan Tomkins algorithm without adaptive thresholding fails to detect some QRS wave in the signal ‘ecg5.dat’ due to presence of high frequency artifact