Assignment 4: Electrical/Bioengineering Case Study

Fear Analysis Using GSR Data

Aman Sunesh (as18181)

STEP 1: Problem Identification and Statement

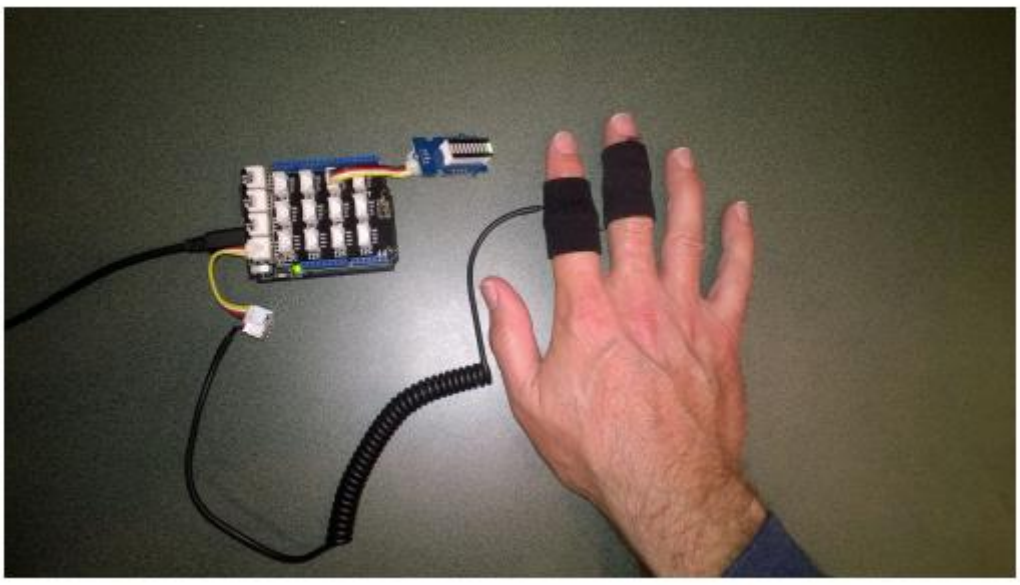
The objective of this assignment is to develop a MATLAB program to calculate the Fear Index by utilizing GSR data from specified data files. The program generates multiple graphs illustrating GSR values (in uS) plotted against time (in seconds). These graphs serve to compare the GSR values both before and after undergoing various filtration processes. Additionally, the program computes and returns the Fear Index, derived from 10 features calculated from the data extracted from the input data files.

STEP 2: Gathering of Information and Input and Output Description

Galvanic Skin Response (GSR) is a measure of the electrical conductance of the skin, influenced by the activity of the sweat glands. It is often used as an indicator of emotional arousal and stress. GSR data is typically recorded in microsiemens (uS) and can be collected using sensors placed on the skin.

The human body's sweating is regulated by the autonomic nervous system. In particular, if the sympathetic branch of the autonomic nervous system is highly aroused, then sweat gland activity also increases, which in turn raises the skin conductance, and vice versa. The galvanic skin response (also called electrodermal response, skin conductance, or psychogalvanic reflex) is the measure of the conductance caused by the variation of the human body sweating.

A GSR sensor allows us to measure sweat gland activity, which is related to emotional arousal. So, to measure GSR, we take advantage of the electrical properties of the skin. Specifically, how the skin resistance changes with sweat gland activity, i.e., the greater sweat gland activity, the more perspiration, and thus, less skin resistance. GSR activity is typically measured in “micro-Siemens (uS)” or “micro-Mho (uM)”, mirroring the conductance of a certain material.



GSR is not only used to monitor your emotional arousal. In psychology, the data from the GSR sensor can be analyse to determine whether a person is lying or not. In fact, the GSR sensor has been widely used in our lives, especially in some medical facilities, psychology, polygraphs, etc.

The Grove GSR sensor is utilized to record the GSR data. The sensor uses two electrodes that are attached to the second and third fingers on one hand. The output of the sensor, an analog voltage signal, is connected to an Arduino microcontroller to convert the data to a digital form using an analog-to-digital converter (ADC). The digital data is then transmitted from the microcontroller to the computer using serial communication. A software running on the computer receives the data and stores it in a data file. Every data file stores the recording of a total of approximately 130 seconds with a sampling frequency of 48 Hz. A snapshot of the sensor is shown below.

10 Features for Fear Index Calculation:

1. **Mean of GSR Signal (F1):**

Description: The average value of the GSR signal over the recording period.

Interpretation: Indicates the central tendency of the skin conductance responses.

2. **Variance of GSR Signal (F2):**

Description: The measure of how much the GSR values deviate from the mean.

Interpretation: Reflects the overall variability or spread of the skin conductance responses.

3. **GSR Peak Rise Time Sum (F3):**

Description: The sum of the rise times for all detected GSR peaks.

Interpretation: Represents the cumulative time it takes for the skin conductance to rise to its peak level.

4. **GSR Peak Amplitude Sum (F4):**

Description: The sum of the amplitudes of all detected GSR peaks.

Interpretation: Indicates the cumulative magnitude of the skin conductance responses.

5. **GSR Peak Energy Sum (F5):**

Description: The sum of the energies of all detected GSR peaks, calculated as 1/2 * amplitude * rise time. Peak Energy Sum = sum(0.5*amplitude*risetime)

Interpretation: Provides a measure that combines both the amplitude and rise time of the skin conductance responses.

6. **Amplitude of the Highest Skin Conductance Response (F6):**

Description: The magnitude of the highest GSR peak in the recording.

Interpretation: Highlights the intensity of the most prominent skin conductance response.

7. **Rise Time of the Highest Skin Conductance Response (F7):**

Description: The time it takes for the GSR signal to rise to its peak for the highest response.

Interpretation: Gives insight into how quickly the most intense skin conductance response occurs.

8. **Number of GSR Peaks (F8):**

Description: The count of distinct peaks in the GSR signal.

Interpretation: Indicates the frequency of skin conductance responses during the recording.

9. **Mean Power of GSR Signal (F9):**

Description: The average power of the GSR signal, computed as the mean of the squared GSR values.

Interpretation: Provides information about the overall strength or intensity of the GSR signal.

10. **GSR Bandwidth (F10):**

Description: The GSR Bandwidth is a measure of the frequency content of the GSR signal. It is calculated using the formula

$$\mathrm{GSR}_{\mathrm{Bwidth}} = \frac{1}{2\pi}\sqrt{\frac{\sum D_{\mathrm{GSR}_i}^2}{\sum \mathrm{GSR}_i^2}}, \quad D_{\mathrm{GSR}_i} = \mathrm{GSR}_i - \mathrm{GSR}_{i-1}$$

, which represents the difference between consecutive GSR values.

Interpretation: GSR Bandwidth provides information about the distribution of frequency components in the GSR signal. A higher GSR Bandwidth may suggest a broader range of changes in skin conductance, reflecting varied physiological responses.

We use all these features to calculate the fear index threshold that indicates the presence of fear using the following equation:

$$FearIndex = (F_1 + 2F_2 + F_3 + 0.5F_4 + F_5 + 2F_6 + F_7 + 5F_8 + 0.001F_9 + 0.5F_{10})$$

Input Description:

The GSR Analyzer program is designed to analyse Galvanic Skin Response (GSR) data, specifically focusing on fear-related physiological responses. The key inputs for the program include the GSR data files, parameters for signal processing filters, and choices for menu navigation.

1. **GSR Data Filenames:**
   - Datafile file (name with extension) containing GSR data are required: 'GSR_FEAR.csv' or 'GSR_Baseline.csv'.

2. **Signal Processing Parameters:**
   - medianOrder: Integer specifying the order of the median filter.
   - N_Order: Integer specifying the order of the N-point moving average filter.
   - Fp: Numeric value specifying the passband frequency for the low-pass filter.

Output Description:

The primary outputs of the GSR Analyzer program are:

1. **Fear Index:**
   - Quantitative measure representing fear-related physiological responses derived from various features of the GSR signal.

2. **Graphs of GSR Values (uS) vs Time (s) (for each datafile):**
   - Before & After Applying 3rd Order Median Filter
   - Before & After Applying Low Pass Filter
   - Before & After an N-point Moving Average Filter
   - Before & After Normalizing the GSR Signal

The I/O diagram for this problem is illustrated below:

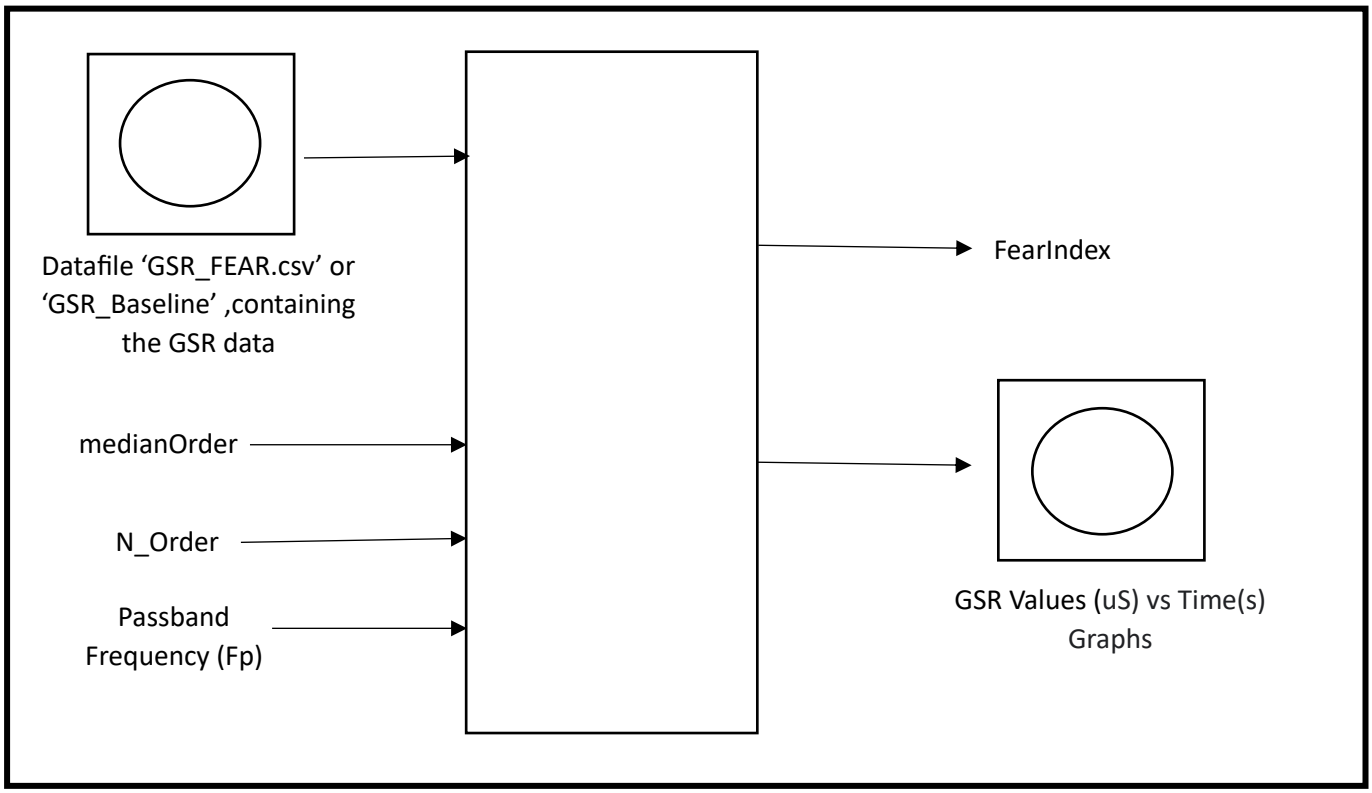


Figure 0.1: I/O Diagram

STEP 3: Test Cases and Algorithm Design

Test Cases: The following dataset is tested with a 3rd order median filter, 10-point moving average filter, and 20Hz passband frequency for the lowpass filter.

1. 'GSR_FEAR.csv' is the datafile that provides the GSR values.

Expected Outputs:

- *The fear index is expected to be higher for this dataset.*
- *Graphs or indicators should show consistent and higher fear levels. The graph should display increased fluctuations and noticeable trough peak separations.*
- *The graphs are expected to exhibit consistent smoothness as a result of the applied filter.*

2. 'GSR_Baseline.csv' is the datafile that provides the GSR values.

Expected Outputs:

- *The fear index is expected to be relatively lower for this dataset.*
- *Graphs or indicators should show fluctuations in fear levels. The graph should display reduced fluctuations and negligible trough peak separations.*
- *The graphs are expected to exhibit a consistent smoothness as a result of the applied filter.*

Algorithm Design:

Main Function:

Assign GSR datafile with extension to filename

Declare and Initialize medianOrder to 3
Declare and Initialize N_Order to 10
Declare and initialize Dp to 20

Call the function loadGSRData(filename) and store the returned value in GSRData

Call the function NPointMovingAvgFilter(N_Order, lowPassFilter(Fp, nthOrderMedianFilter(medianOrder, GSRData))) and store the returned value in filteredGSRData

Call the function normalizeGSRSignal(filteredGSRData) and store the returned value in normalizedGSRData

Call the function GSRFeatures(normalizedGSRData) and store the returned value in F

Calculate FearIndex by calling calcFearIndex(F) and store the result in FearIndex
Print 'The fear index is: ', FearIndex

End Main

Define function loadGSRData(filename)

Open file filename in read mode

If file open is unsuccessful
Print 'Error! File open not successful'
End If

Read the file header

Read data from file in the required format and store the data in the variable data

Close the file fid and store the returned value in closeresult

If closeresult is not equal to 0
Print 'File close not successful'
End If

```
        Declare and Initialize time to seconds(data{1} – data{1}(1))

        Declare and Initialize a cell array GSRData to {time, data{2}}

        Return GSRData

End Function



Define function nthOrderMedianFilter(N, GSRData)
        Declare and Initialize filteredGSRData to GSRData

        Declare and Initialize i to 1
        Repeat while i<= length(GSRData{2})
                Declare and Initialize subset to array of zeros with size N

                Declare and Initialize k to 1

                Declare and Initialize j to -floor((N-1)/2)
                Repeat while j <= ceil((N-1)/2)
                   Declare and Initialize index to i+j
                   Set index to max(1, min(index, length(GSRData{2})))
                  Set subset(k) to GSRData{2}(index)
                  Increment k by 1
                End for

                Set filteredGSRData{2}(i) to median(subset)
          End for

          Create a new figure

          Create subplot with 2 rows and 1 column, activate the first subplot
          Plot GSRData{1} versus GSRData{2}
          Set title to 'Time vs GSR Values Before nth Order Median Filtration'
          Set xlabel to 'Time (s)'
          Set ylabel to 'GSR Values (uS)'

          Activate the second subplot
         Plot filteredGSRData{1} versus filteredGSRData{2} in red
         Set title to 'Time vs Filtered GSR Values After nth Order Median Filtration'
         Set xlabel to 'Time (s)'
         Set ylabel to 'Filtered GSR Values (in uS)'

        Save the figure as 'nthOrderMedianFilter.jpg'

       Return filteredGSRData

End function
```

```
Define function lowPassFilter(Fp, GSRData)
        Declare and Initialize filteredGSRData to GSRData

        Declare and Initialize timeDif to diff(GSRData{1})

        Declare and Initialize mWS to 1/Fp

        Declare and Initialize meanWindowPoints to round(mWS / median(timeDif))

        Set filteredGSRData{2} to smoothdata(GSRData{2}, 'movmean', meanWindowPoints)

        Create a new figure

        Create subplot with 2 rows and 1 column, activate the first subplot
        Plot GSRData{1} versus GSRData{2}
        Set title to 'Time vs GSR Values Before Low Pass Filtration'
        Set xlabel to 'Time (s)'
        Set ylabel to 'GSR Values (uS)'

       Activate the second subplot
      Plot filteredGSRData{1} versus filteredGSRData{2} in red
      Set title to 'Time vs Filtered GSR Values After Low Pass Filtration'
      Set xlabel to 'Time (s)'
     Set ylabel to 'Filtered GSR Values (in uS)'

     Save the figure as 'LowpassFilter.jpg'

    Return filteredGSRData

End function



Define function NPointMovingAvgFilter(N, GSRData)
        Declare and Initialize filteredGSRData to GSRData

        Declare and Initialize i to 0
        Repeat while i <= length(GSRData{2})
                Declare and Initialize subset to array of zeros with size N

                Declare and Initialize k to 1

                Declare and Initialize j to -floor((N-1)/2)
                Repeat while j <= ceil((N-1)/2)
                        Declare and Initialize index to i+j
                        Set index to max(1, min(index, length(GSRData{2})))
                        Set subset(k) to GSRData{2}(index)
                        Increment k by 1
                End for

                Set filteredGSRData{2}(i) to mean(subset)
          End for

          Create a new figure

          Create subplot with 2 rows and 1 column, activate the first subplot
          Plot GSRData{1} versus GSRData{2}
          Set title to 'Time vs GSR Values Before Applying N-Point Moving Average Filtration'
```

```
        Set xlabel to 'Time (s)'
        Set ylabel to 'GSR Values (uS)'

        Activate the second subplot
        Plot filteredGSRData{1} versus filteredGSRData{2} in red
        Set title to 'Time vs Filtered GSR Values After Applying N-Point Moving Average Filtration'
       Set xlabel to 'Time (s)'
       Set ylabel to 'Filtered GSR Values (in uS)'

        Save the figure as 'NPointMovingAvgFilter.jpg'

    Return filteredGSRData

End function



Define function normalizeGSRSignal(GSRData)
    Declare and Initialize normalizedData to GSRData

    Declare and Initialize i = 1
    Repeat while i <= length(GSRData{2})

        Declare and Initialize minGSR to the minimum value of GSRData{2}
        Declare and Initialize maxGSR to the maximum value of GSRData{2}

        Declare and Initialize normalizedGSR to ((GSRData{2}(i) - minGSR) / (maxGSR - minGSR)) * 100

        Set normalizedData{2}(i) to normalizedGSR
    End for

    Create a new figure

    Create subplot with 2 rows and 1 column, activate the first subplot
    Plot GSRData{1} versus GSRData{2}
    Set title to 'Time vs GSR Values Before Normalization'
    Set xlabel to 'Time (s)'
    Set ylabel to 'GSR Values (uS)'

    Activate the second subplot
    Plot normalizedData{1} versus normalizedData{2} in red
    Set title to 'Time vs Filtered GSR Values After Normalization'
    Set xlabel to 'Time (s)'
    Set ylabel to 'Filtered GSR Values (in uS)'

    Save the figure as 'Normalised.jpg'

    Return normalizedData

End function
```

```
Define function GSRFeatures(GSRData)
        Set F(1) to mean(GSRData{2})
        Set F(2) to var(GSRData{2})

        Set max_peak to islocalmax(GSRData{2})
        Set min_peak to islocalmin(GSRData{2})

        Set maxima to GSRData{2}(max_peak)
        Set minima to GSRData{2}(min_peak)
        Set maxTime to GSRData{1}(max_peak)
        Set minTime to GSRData{1}(min_peak)

        Set tr_pk_sep_prctl to 16.305

        Set size to length(maxima)
        Create amplitude array with size elements, initialized to zeros
        Create riseTime array with size elements, initialized to zeros
        Set F(3) to 0
        Set F(5) to 0


        Declare and Initialize i = 0
        Repeat while i <= size
           Set index to find(minTime < maxTime(i), 1, 'last')

           If  the variable index is not empty
              If ((maxima(i) - minima(index)) > tr_pk_sep_prctl)
                 Set amplitude(i) to maxima(i)
                 Set riseTime(i) to maxTime(i) - minTime(index)

                 Set F(3) to F(3) + riseTime(i)

                 Set F(5) to F(5) + (0.5 * amplitude(i) * riseTime(i))
              End if
           End if
        End for

        Set F(4) to sum(amplitude)

        Set [F(6), index] to max(amplitude)

        Set F(7) to riseTime(index)

        Set F(8) to nnz(amplitude)

        Set F(9) to mean(GSRData{2}.^2)

        Set D_GSR to diff(GSRData{2})
        Set D_GSR_2 to sum(D_GSR.^2)
        Set sumGSR2 to sum(GSRData{2}.^2)
        Set GSR_Bandwidth to (1/(2*pi)) * sqrt(D_GSR_2/sumGSR2)

        Set F(10) to GSR_Bandwidth

       Return F


End function
```

Define function calcFearIndex(F)

*Set FearIndex to F(1) + (2*F(2)) + F(3) + (0.5*F(4)) + F(5) + (2*F(6)) + F(7) + (5*F(8)) + (0.001*F(9)) + (0.5*F(10))*

Return FearIndex

End function

STEP 4: Implementation

```
%% Main function to process GSR data and calculate Fear Index
function main
    % Define input parameters
    filename = 'GSR_FEAR.csv';    % Input GSR data file
    medianOrder = 3;              % Order for median filter
    N_Order = 10;                 % Order for nth order median filter
    Fp = 20;                      % Passband frequency for low-pass filter


    % Load GSR data from file
    GSRData = loadGSRData(filename);

    % Apply filters to the GSR data
    filteredGSRData = NPointMovingAvgFilter(N_Order, lowPassFilter(Fp,
nthOrderMedianFilter(medianOrder, GSRData)));

    % Normalize the filtered GSR data
    normalizedGSRData = normalizeGSRSignal(filteredGSRData);

    % Determine features about the GSR signal
    F = GSRFeatures(normalizedGSRData);

    % Calculate Fear Index based on the extracted features
    FearIndex = calcFearIndex(F);

    % Calculate Fear Index based on the extracted features
    fprintf('The fear index is: %.2f \n', FearIndex)
end



%% loadGSRData function to read GSR data from a file and process it
function GSRData  = loadGSRData(filename)
    % Open the file for reading
    fid = fopen(filename,'r');

    % Check if the file open operation is successful
    if (fid == -1)
        error('Error! File open not successful');
    end

    %Reads the file header
```

```
    fscanf(fid, '%s', 2);

    % Read data from the file using textscan
    data = textscan(fid, '%{mm:ss:SSS}D %f', 'Delimiter',',');

    closeresult = fclose(fid);

    % Check if the file close operation is unsuccessful
    if closeresult ~= 0
        disp('File close not successful');
    end

    %Converting time to seconds by offsetting all values by the first time
    %reading
    time = seconds(data{1} - data{1}(1));

    % Organize the data into a cell array containing time and GSR values
    GSRData = {time,data{2}};
end



%% nthOrderMedianFilter function to apply median filtering to GSR data
function filteredGSRData = nthOrderMedianFilter(N, GSRData)

    % Initialize filteredGSRData with the input GSRData
    filteredGSRData = GSRData;

    % Loop through each GSR data point
    for i = 1:length(GSRData{2})

        % Create a subset for median filtering
        subset = zeros(1,N);

        k = 1;

        % Iterate through the neighborhood of the current data point
        for j = -floor((N-1)/2):ceil((N-1)/2)
            index = i+j;
            index = max(1,min(index, length(GSRData{2})));
            subset(k) = GSRData{2}(index);
            k = k+1;
        end

        % Apply median filtering to the subset and update filteredGSRData
        filteredGSRData{2}(i) = median(subset);

    end


    % Plot the original and filtered GSR data

    figure;
    subplot(2,1,1);
    plot(GSRData{1},GSRData{2});
    title('Time vs GSR Values Before nth Order Median Filtration');
    xlabel('Time (s)');
```

```
    ylabel('GSR Values (uS)');

    subplot(2,1,2);
    plot(filteredGSRData{1},filteredGSRData{2}, 'r');
    title('Time vs Filtered GSR Values After nth Order Median Filtration');
    xlabel('Time (s)');
    ylabel('Filtered GSR Values (in uS)');

    % Save the figure as a jpg file
    saveas(gcf, 'nthOrderMedianFilter', 'jpg');

end



%% lowPassFilter function to apply low-pass filtering to GSR data
function filteredGSRData = lowPassFilter(Fp, GSRData)

    % Initialize filteredGSRData with the input GSRData
    filteredGSRData = GSRData;

    % Calculate the time differences between consecutive data points
    timeDif = diff(GSRData{1});

    % Calculate the mean window size based on the specified passband frequency Fp
    mWS = 1/Fp; %where mWS represents the mean window size

    % Calculate the number of points for the moving mean window
    meanWindowPoints = round(mWS / median(timeDif));

    % Apply moving mean (low-pass) filtering to GSR data
    filteredGSRData{2} = smoothdata(GSRData{2}, 'movmean', meanWindowPoints);


    % Plot the original and filtered GSR data

    figure
    subplot(2,1,1);
    plot(GSRData{1},GSRData{2});
    title('Time vs GSR Values Before Low Pass Filtration');
    xlabel('Time (s)');
    ylabel('GSR Values (uS)');

    subplot(2,1,2);
    plot(filteredGSRData{1},filteredGSRData{2}, 'r');
    title('Time vs Filtered GSR Values After Low Pass Filtration');
    xlabel('Time (s)');
    ylabel('Filtered GSR Values (in uS)');

    % Save the figure as a jpg file
    saveas(gcf, 'LowpassFilter', 'jpg');

end
```

```
%% NPointMovingAvgFilter function to apply N-Point Moving Average filtering
to GSR data
function filteredGSRData = NPointMovingAvgFilter(N, GSRData)

    % Initialize filteredGSRData with the input GSRData
    filteredGSRData = GSRData;

    % Iterate through each data point in GSRData
    for i = 1:length(GSRData{2})

        % Initialize a subset array to store N data points for averaging
        subset = zeros(1,N);

        % Initialize index variables for creating the subset
        k = 1;

        % Create the subset based on the specified window size N
        for j = -floor((N-1)/2):ceil((N-1)/2)
            index = i+j;
            index = max(1,min(index, length(GSRData{2})));
            subset(k) = GSRData{2}(index);
            k = k+1;
        end

        % Replace the current data point with the mean of the subset
        filteredGSRData{2}(i) = mean(subset);
    end

    % Plot the original and filtered GSR data

    figure;
    subplot(2,1,1);
    plot(GSRData{1},GSRData{2});
    title('Time vs GSR Values Before Applying N-Point Moving Average
Filtration');
    xlabel('Time (s)');
    ylabel('GSR Values (uS)');

    subplot(2,1,2);
    plot(filteredGSRData{1},filteredGSRData{2}, 'r');
    title('Time vs Filtered GSR Values After Applying N-Point Moving
Average Filtration');
    xlabel('Time (s)');
    ylabel('Filtered GSR Values (in uS)');

    % Save the figure as a jpg file
    saveas(gcf, 'NPointMovingAvgFilter', 'jpg');
end
```

```
%% normalizeGSRSignal function to normalize GSR data
function normalizedData = normalizeGSRSignal(GSRData)

    % normalizeGSRSignal function to normalize GSR data
    normalizedData = GSRData;

    % Iterate through each data point in GSRData
    for i = 1:length(GSRData{2})
        minGSR = min(GSRData{2});
        maxGSR = max(GSRData{2});

        % Normalize the GSR value using the min-max normalization formula
        normalizedGSR = ((GSRData{2}(i) - minGSR) / (maxGSR - minGSR))*
        100;

        % Update the normalized GSR value in the normalizedData array
        normalizedData{2}(i) = normalizedGSR;
    end

    % Plot the original and normalized GSR data

    figure;
    subplot(2,1,1);
    plot(GSRData{1},GSRData{2});
    title('Time vs GSR Values Before Normalization');
    xlabel('Time (s)');
    ylabel('GSR Values (uS)');

    subplot(2,1,2);
    plot(normalizedData{1},normalizedData{2}, 'r');
    title('Time vs Filtered GSR Values After Normalization');
    xlabel('Time (s)');
    ylabel('Filtered GSR Values (in uS)');

    % Save the figure as a jpg file
    saveas(gcf, 'Normalised', 'jpg');

end



%% GSRFeatures function to extract features from GSR data
function F = GSRFeatures(GSRData)

    % Feature 1: Mean of GSR values
    F(1) = mean(GSRData{2});

    % Feature 2: Variance of GSR values
    F(2) = var(GSRData{2});

    % Identify local maxima and minima in GSR data
    max_peak = islocalmax(GSRData{2});
    min_peak = islocalmin(GSRData{2});

    % Extract maxima and minima values along with corresponding times
    maxima = GSRData{2}(max_peak);
    minima = GSRData{2}(min_peak);
```

```
    maxTime = GSRData{1}(max_peak);
    minTime = GSRData{1}(min_peak);

    % Trough-Peak seperation threshold calculated using GSR_FEAR.csv data
    %trough-peak seperation values at 90% percentile
    tr_pk_sep_prctl = 16.305;

    % Initialize variables for amplitude, rise time, and cumulative
features
    size = length(maxima);
    amplitude = zeros(1,size);
    riseTime = zeros(1,size);
    F(3) = 0;
    F(5) = 0;



    % Initialize variables for amplitude, rise time, and cumulative
features
    for i = 1:size
        index = find(minTime < maxTime(i), 1, 'last');

        if ~isempty(index)

            % Initialize variables for amplitude, rise time, and cumulative
            features
            if ((maxima(i) - minima(index)) > tr_pk_sep_prctl)

                amplitude(i) = maxima(i);

                riseTime(i) = maxTime(i) - minTime(index);

                % Feature 3: GSR Peak Rise Time Sum
                F(3) = F(3) + riseTime(i);

                % Feature 5: GSR Peak Energy Sum
                F(5) = F(5) + (0.5 * amplitude(i) * riseTime(i));
            end
        end
    end


    % Feature 4: GSR Peak Amplitude Sum
    F(4) = sum(amplitude);

    % Feature 6: Amplitude of the highest skin conductance response
    [F(6), index] = max(amplitude);

    % Feature 7: Rise time of the highest skin conductance response
    F(7) = riseTime(index);

    % Feature 8: Number of non-zero amplitudes
    F(8) = nnz(amplitude);

    % Feature 9: Mean power of GSR signal
    F(9) = mean(GSRData{2}.^2);
```

```
    % Calculate differences and bandwidth for Feature 10
    D_GSR = diff(GSRData{2});

    D_GSR_2 = sum(D_GSR.^2);

    sumGSR2 = sum(GSRData{2}.^2);

    GSR_Bandwith = (1/(2*pi)) * sqrt(D_GSR_2/sumGSR2);

    % Feature 10: GSR Bandwidth
    F(10) = GSR_Bandwith;

end
```

```
%% Function calcFearIndex calculates the Fear Index based on a weighted sum
of extracted features (F).
function FearIndex = calcFearIndex(F)
      FearIndex = F(1) + (2*F(2)) + F(3) + (0.5*F(4)) + F(5) + (2*F(6)) +
      F(7) + (5*F(8)) + (0.001*F(9)) + (0.5*F(10));
end
```

STEP 5: Test and Verification (and Debugging)

Test Case 1: 'GSR_FEAR.csv' is the datafile that provides the GSR values.

The fear index is: 2571.59

Conclusions:

- The fear index is indeed higher for this dataset, suggesting elevated levels of fear.
- Graphs and indicators exhibit consistent and higher fear levels with increased fluctuations and noticeable trough-peak separations.
- The applied filter maintains the consistency and smoothness of the graphs, enhancing the clarity of fear-related patterns in the data.

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Test Case 2: 'GSR_Baseline.csv' is the datafile that provides the GSR values.

The fear index is: 2251.19

Conclusions:

- The fear index is indeed relatively lower for this dataset, indicating baseline or lower levels of fear.
- Graphs and indicators show fluctuations in fear levels, with reduced fluctuations and negligible trough-peak separations.
- The applied filter successfully maintains consistent smoothness in the graphs, preserving clarity in fear-related patterns even at lower levels.

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Clearly, from test cases 1 & 2, it is visible that the value of the fear index is higher when 'GSR_FEAR.csv' is the datafile that provides the GSR values as compared to when 'GSR_Baseline.csv' is the datafile that provides the GSR values (as expected).

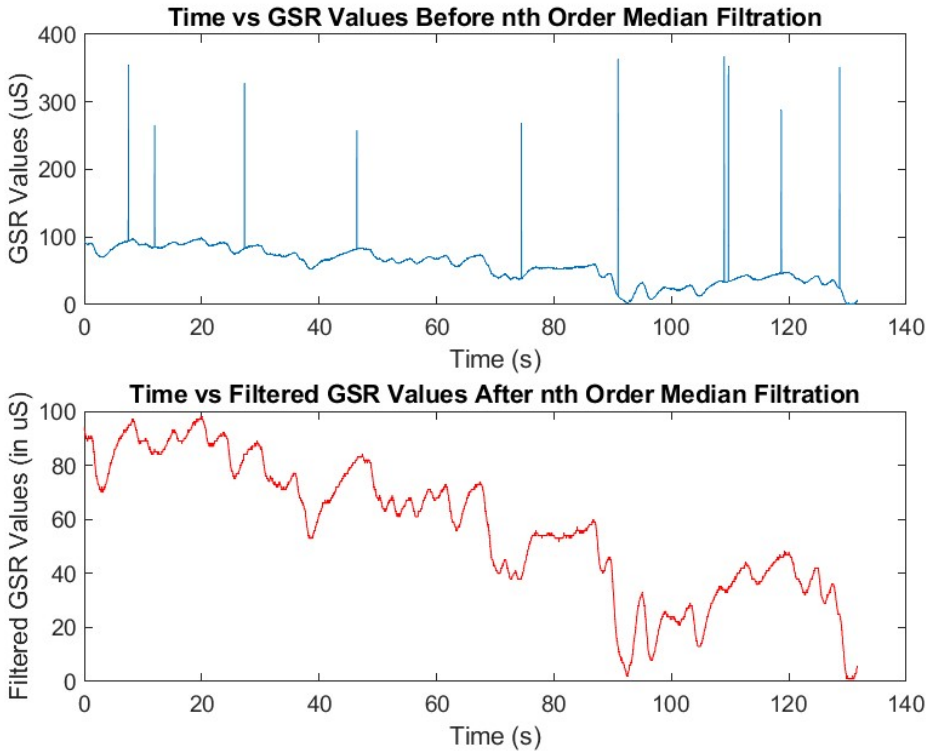
,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

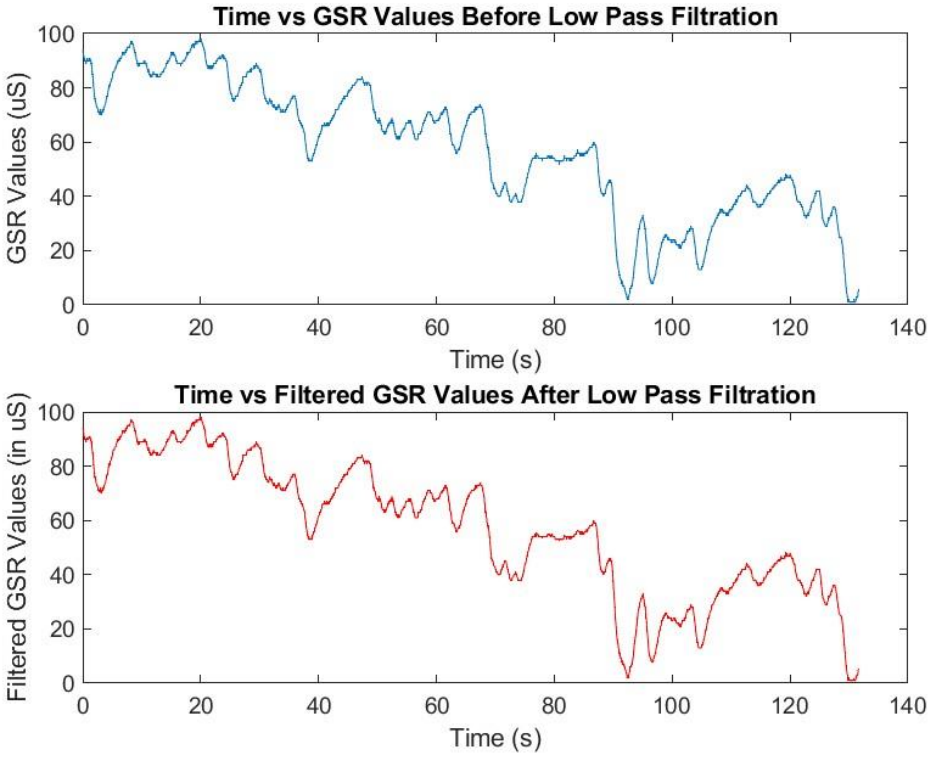
Sample Outputs –

1) GSR_FEAR.csv

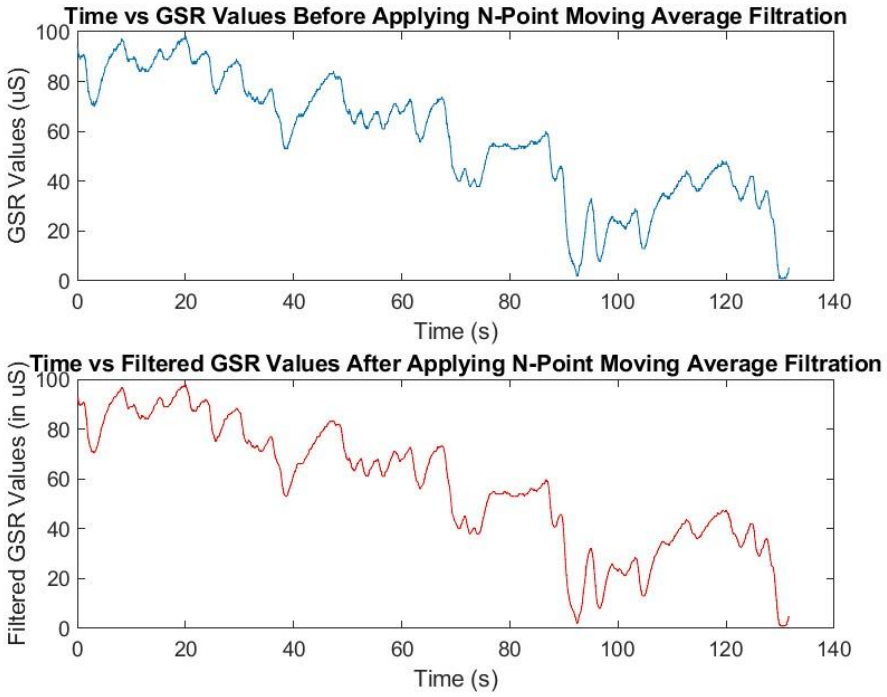
a)



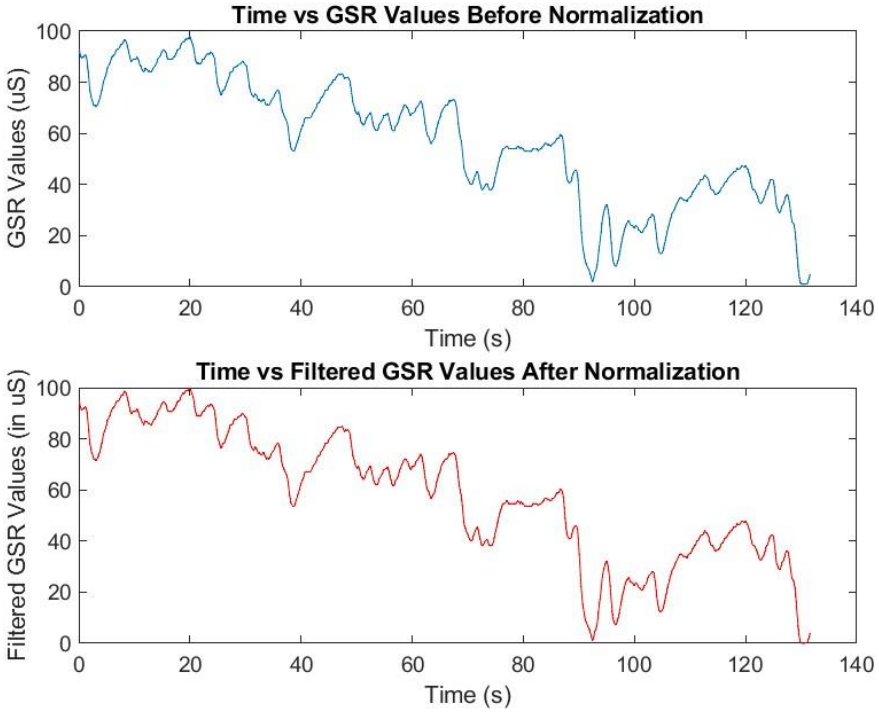
b)



c)



d)

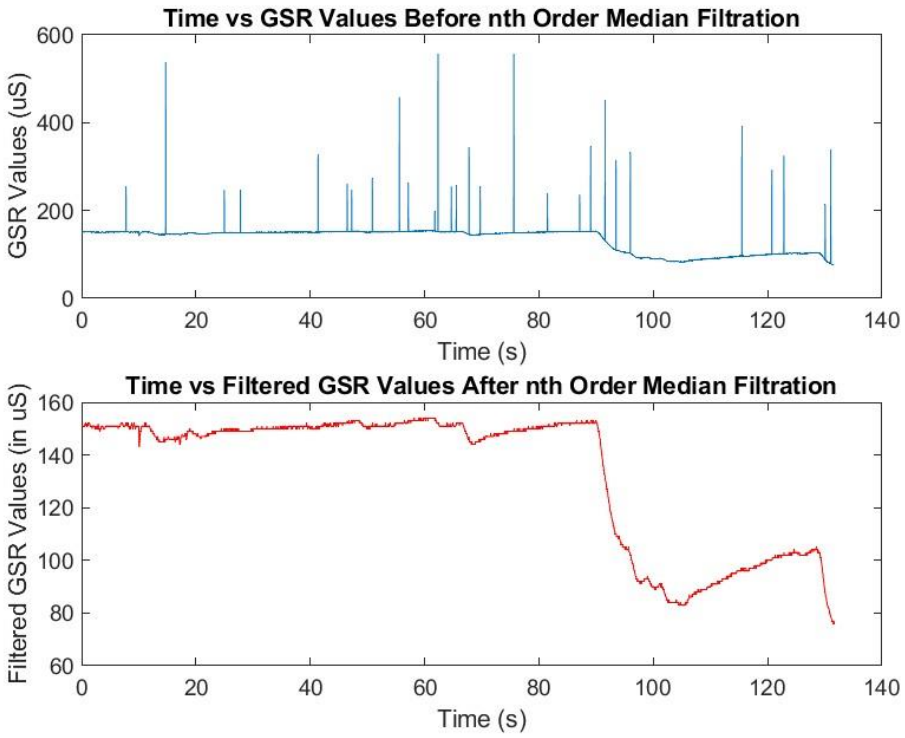



```
>> main
The fear index is: 2571.59
```

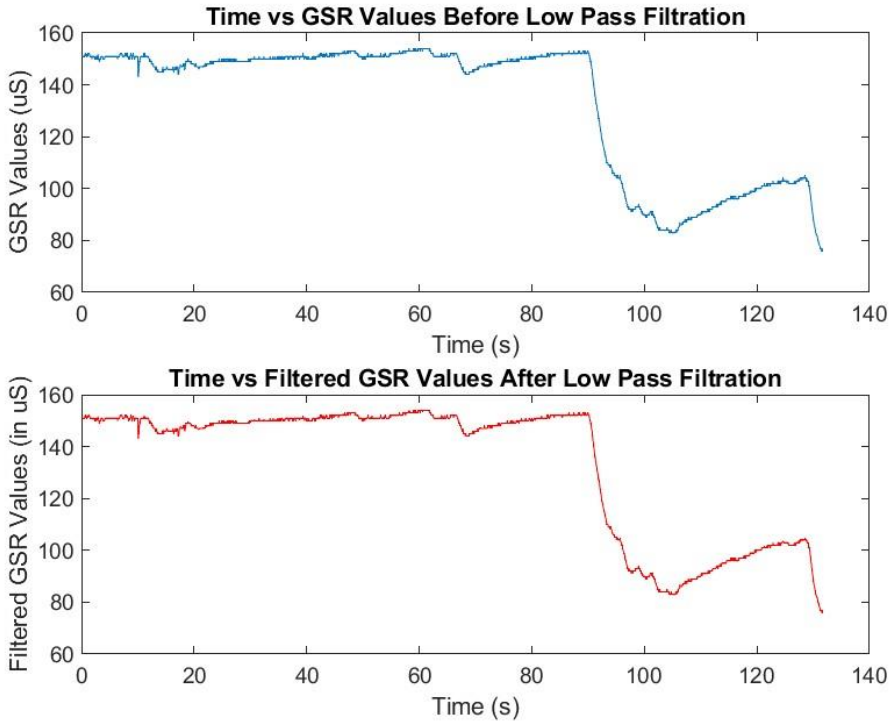
,which is in agreement with the test case expected output.
Therefore, we can conclude that the program is functioning correctly.

2) GSR_Baseline.csv

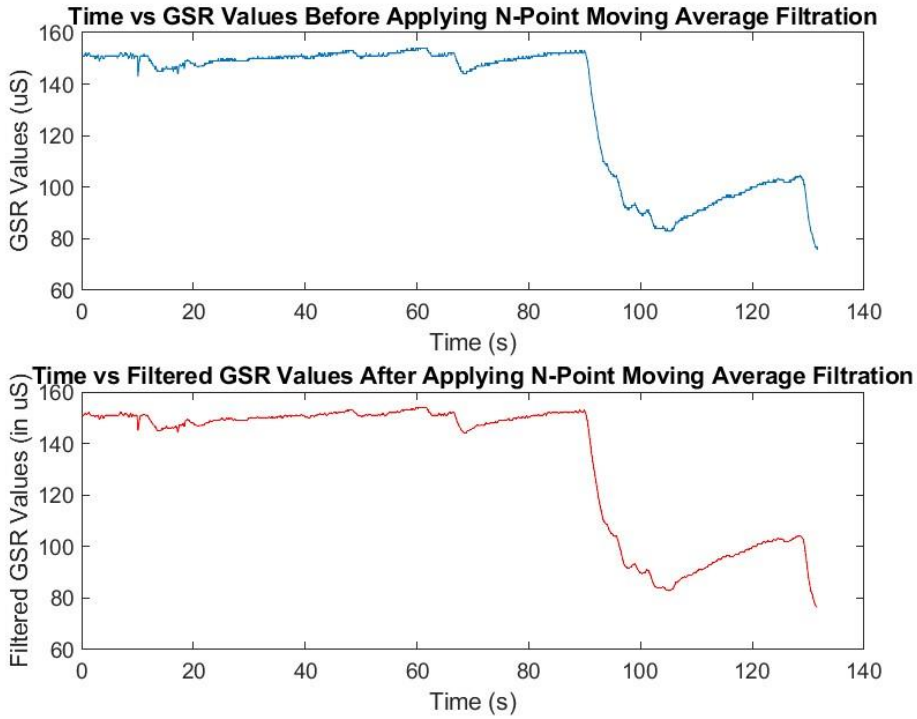
a)



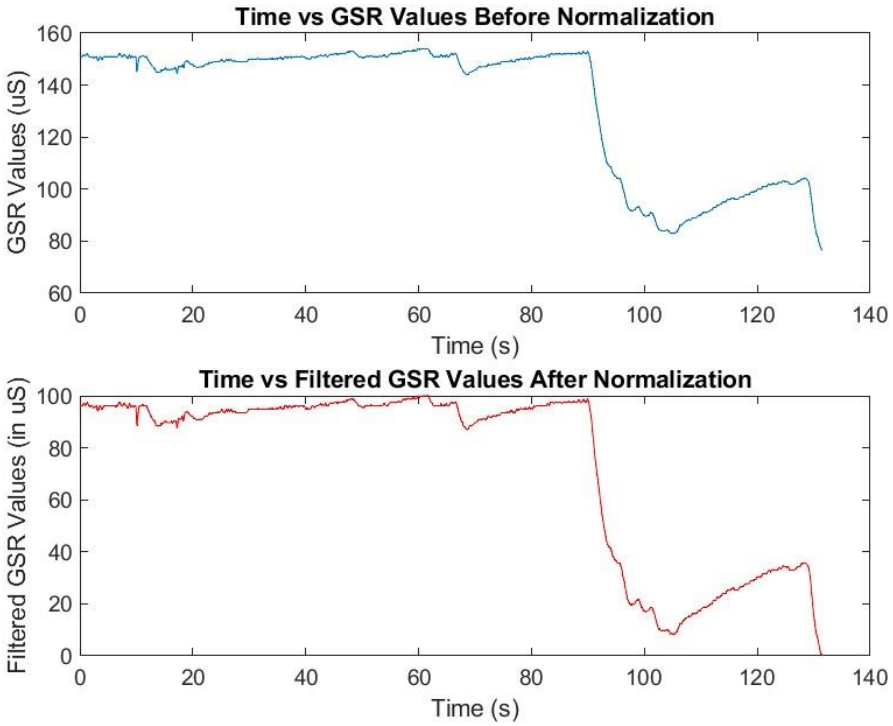
b)



c)



d)




```
>> main
The fear index is: 2251.19
```

,which is in agreement with the test case expected output.
Therefore, we can conclude that the program is functioning correctly.

User Guide:

- To execute the program, compile and run the code found in the file named main.m
- The user can change the values of the input parameters by modifying the values of the variables found in the main function.
- The user may have re-run the program each time they wish to analyse a new file.

1. Loading Data

- Load GSR Data
  - Use the loadGSRData function to load fear and baseline recordings.
  - Provide file paths for fear and baseline data as prompted.

2. Signal Processing

- 2. 1 Apply 3rd Order Median Filter
  - Utilize the thirdOrderMedianFilter function to remove spikes from raw data.
  - Observe the data plots before and after filtration.

- 2.2 Apply Low Pass Filter
  - Apply a low pass filter with a passband frequency of 20 Hz using the lowPassFilter function.
  - Examine data plots before and after filtration.

- 2.3 Apply Moving Average Filter
  - Use the movingAverageFilter function to apply a 10-point moving average filter.
  - Review data plots before and after filtration.

- 2.4 Normalize GSR Signal
  - Normalize the GSR signal between 0 and 100 with the normalizeGSR function.
  - Inspect plots before and after normalization.

3. Feature Extraction

- 3.1 Extract Features
  - Extract 10 features using the extractGSRFeatures function.
  - Features include mean, variance, peak rise time sum, peak amplitude sum, peak energy sum, highest response amplitude, highest response rise time, number of peaks, and mean power.

4. Fear Index Calculation

- 4.1 Calculate Fear Index
  - Use the calculateFearIndex function to determine the fear index based on extracted features.
  - Set a suitable threshold to indicate the presence of fear.

Google Drive Backup: https://drive.google.com/drive/folders/12xNS3DavmcCso6THiakYGHai1-O6tVKz?usp=drive_link