ELEC431
Engineering Programming

Assignment 1 -
Graphic User Interface (GUI)

Khanthapak Thaipakdee

Department of Electrical Engineering and Electronics,
University of Liverpool,
Brownlow Hill, Liverpool L69 3GJ, UK

Table of Contents

List of Figures

List of Tables

1.Introduction and Specification Analysis

1.1 Introduction

This assignment serves as a practical avenue for gaining hands-on experience in software development. It encompasses crucial aspects such as specification analysis, program structure design, coding, testing, program integration, and documentation.

In this assignment, it is imperative to develop a MATLAB GUI program designed to extract coordinate values from a text file. and perform the necessary processing. The program should exhibit the following functionalities:

- Prompt the user to specify the file name from which the data will be retrieved.
- Graphically represent two time-resolved electrical signals, namely the high voltage signal "Uapp" extracted from the "Uapp.txt" file and the charge "Q," extracted from the "Uc.txt" with the x-axis and y-axis.
- Calculate various parameters for both signals, including peak-to-peak amplitude, root mean square (RMS) value, frequency, minimum value, and maximum value.
- Illustrate the Lissajous figure (Q-U) and apply a smoothing operation.
- Compute and present the discharge power.

1.2 Specification Analysis

1.2.1 Input Analysis

1) Retrieve the path and filename of the data file to be opened from mouse.

Note: The dialogue box will exclusively exhibit ".txt" files. User can cancel the dialogue box without receiving any error. It will display “User selected Cancel” via command window.

2) Specify the options for graph display: choose between displaying smoothed data or non-smoothed data.

1.2.2 Output Analysis

1) Provide the path and filename of the opened data file. The filename will be visible on the GUI; if the user opts not to select any file, no graph will be displayed and a warning message, "User selected Cancel," will be shown via the command window.
2) Display the graph based on the chosen option: smoothed data or non-smoothed data. If the user selects smoothed data, the graph will be presented in its smoothed version.

1.2.3 List of major tasks

1) Reading Data from Files: The initial step involves extracting data from the files.
2) Determining Parameters of the Data: Subsequently, an analysis is conducted to identify key parameters associated with the data.
3) Displaying Data: The acquired data is then presented visually, ensuring effective representation.

4) Smoothing Data: As part of data preprocessing, a smoothing process is applied to enhance the clarity and interpretability of the information.
5) Power Calculation: Finally, a computation is performed to ascertain the power associated with the dataset, providing additional insights into its characteristics.

2.Program Design

2.1 Hierarchy Chart

The main program integrates various modules to fulfill all specified requirements. Sub-function 1, named importdata, is responsible for importing data from the selected files. Sub-function 2, titled filter_data, employs down-sampling and a moving-average filter to process the input data. Sub-function 3, denoted as find_details, identifies essential parameters within the dataset. Sub-function 4, named power_calculation, computes and presents the discharge power. Finally, Sub-function 5, labeled plotting_graphs, is dedicated to graph plotting, visualizing the acquired data.

This structure ensures a systematic and organized approach to the implementation of each module, contributing to the overall effectiveness of the program. Hierarchy chart is Figure 1.

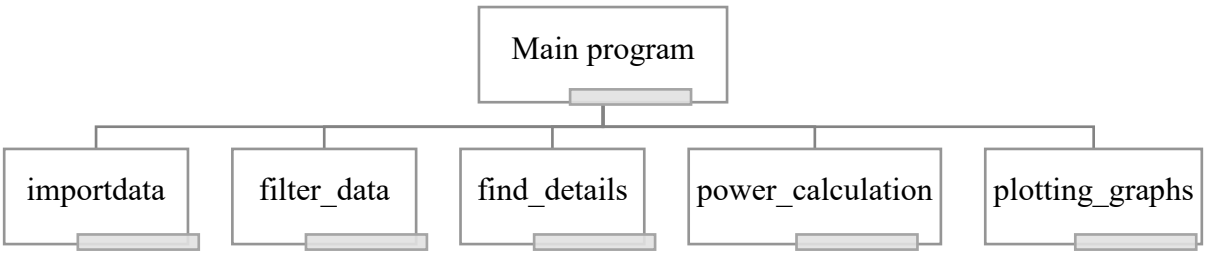

Figure 1: Hierarchy chart

2.2 Data Table

2.2.1 Properties of the program

Table 1: Properties of the program

Property name	Data type	Description
uapp_x	Double	Time of Uapp from the imported file
uapp_y	Double	Value of Uapp from the imported file
q_x	Double	Time of Uc from the imported file
q_y	Double	Value of Uc from the imported file multiply with 22 nF
t_config	Double	Time after filtering
uapp_yfilter	Double	Value of Uapp after filtering
q_yfilter	Double	Value of Q after filtering
locs_Maxvolt	Double	Index of the highest point of each cycle of Uapp
locs_Maxcharge	Double	Index of the highest point of each cycle of Q
locs_Minvolt	Double	Index of the lowest point of each cycle of Uapp
locs_Mincharge	Double	Index of the lowest point of each cycle of Q

2.2.2 Data Table for sub_function_1 (importdata)

Data table of importdata sub-function is Table 2.

Table 2: Data Table for “importdata”

Variable name	Data type	Description
filename	Char	The input file name
T	Table	The input data (200000 x 2 array)

2.2.3 Data Table for sub_function_2 (filter_data)

Data table of filter_data sub-function is Table 3.

Table 3: Data Table for “filter_data”

Variable name	Data type	Description
xvalue	Double	Input parameter of function (filter_data) refers to the time of Uapp/Q
yvalue	Double	Input parameter of function (filter_data) refers to the value of Uapp/Q
xfilter	Double	Output parameter of function (filter_data) refers to the time of Uapp/Q after configuration
yfilter	Double	Output parameter of function (filter_data) refers to the value of Uapp/Q after filtering
m	Double	Size of the signal
down_size	Double	Size of down sampling
moving_size	Double	Size of moving average filter
signal_ydown	Double	Value of yvalue after down sampling
signal_ymov	Double	Value of signal_ydown after passing moving average filter
signal_xdown	Double	Value of xvalue after down sampling
signal_xmov	Double	Value of signal_xdown after passing moving average filter

2.2.4 Data Table for sub_function_3 (find_details)

Data table of find_details sub-function is Table 4.

Table 4: Data Table for “find_details”

Variable name	Data type	Description
xvalue	Double	Input parameter of function (find_details) refers to the time of Uapp/Q
yvalue	Double	Input parameter of function (find_details) refers to the value of Uapp/Q
min_value	Double	Minimum value of yvalue (Uapp/Q)
max_value	Double	Maximum value of yvalue (Uapp/Q)
p2p_value	Double	Average peak-to-peak value of yvalue (Uapp/Q)
rms_value	Double	Root means square value of yvalue (Uapp/Q)
peak	Double	Value of highest point of each peak
invpeak	Double	Value of lowest point of each peak
f_value	Double	Average frequency of Uapp/Q
n	Double	Number of the cycle of imported data
f	Double	Summation of frequency from each cycle
p2p_u	Double	Summation of peak-to-peak value from each cycle

2.2.5 Data Table for sub_function_4 (power_calculation)

Data table of power_calculation sub-function is Table 5.

Table 5: Data Table for “power_calculation”

Variable name	Data type	Description
power_value	Double	Average power of each cycle
m	Double	Size of index between each cycle (use to create “line”)
n	Double	Number of the cycle of imported data
i	Double	Index for “for loop”
j	Double	Index for “for loop”
k	Double	Index of “line”
locstart	Double	Start index of each cycle
locstop	Double	Stop index of each cycle
line	Double	Value of Uapp and Q in each cycle
power	Double	Summation of power in each cycle

2.2.6 Data Table for sub_function_5 (plotting_graphs)

Data table of plotting_graphs sub-function is Table 6.

Table 6: Data Table for “plotting_graphs”

Variable name	Data type	Description
uapp_yfilter	Double	Value of Uapp after filtering
q_yfilter	Double	Value of Q after filtering
t_config	Double	Time after filtering

3. GUI Design and Program Testing

3.1 GUI Design

Figure 2 illustrates the GUI layout of the program. Separate graphs for the Uapp signal and the Q signal are plotted at the top of the layout, each accompanied by their respective signal parameters. The Lissajous figure visually represents the correlation between charge on the x-axis and voltage on the y-axis. Below the Lissajous figure is the power section. The software's title is positioned in the left panel of the design, while an additional tab, located on the right panel, provides detailed information regarding the signals.

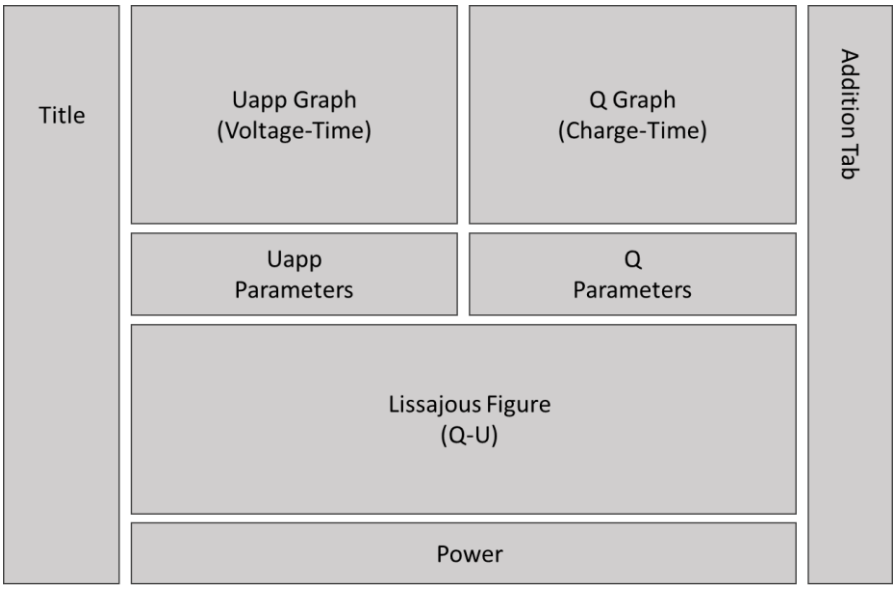

Figure 2: GUI layout

3.2 Functions

3.2.1 Callback functions: ImportButton_UappPushed, ImportButton_UcPushed

Callback functions, namely “ImportButton_UappPushed” and “ImportButton_UcPushed”, will be invoked when the user presses the “ImportButton_Uapp” button or “ImportButton_Uc” button, respectively.

The "uigetfile" function was used to open a dialogue box that lists files in the current folder, enabling the user to select a file by specifying '*.txt' as the filter input argument. An "if-else" condition was implemented to verify that the user selects a file, thus preventing errors. Following this, the "readtable" function was employed to import the contents of a text file into a table.

Subsequently, the first column's data was stored as the property "uapp_x," and the second column's data was stored as the property "uapp_y." A graph was then plotted between "uapp_x" and "uapp_y" with specific attributes. This process was followed by the use of "filter_data" and "find_details" functions, responsible for filtering data and displaying signal parameters on the app screen.

Similarly, if the "ImportButton_Uc" button is triggered, it will perform the same tasks, with the distinction that the data from the first column will be stored as the property "q_x," and the data from the second column, multiplied by $22x10^{(-9)}$, will be stored as the property "q_y."

```
[filename,] = uigetfile('*.txt');

if isequal(filename,0)
   disp('User selected Cancel');

else
   %% importdata %%
   app.FilenameUapp.Value=filename;
   T=readtable(filename);

   app.uapp_x = table2array(T(:,1));
   app.uapp_y = table2array(T(:,2));

   % graph plotting%
   % filter_data function %
   % find_details function %
```

```
%% importdata %%
 app.FilenameUc.Value=filename;
 T=readtable(filename);
 app.q_x = table2array(T(:,1));
 app.q_y = 22*1e-9.*table2array(T(:,2));
```

3.2.2 Callback function: LissajousPlotButtonPushed

Callback functions, specifically "LissajousPlotButtonPushed," will be triggered upon the user pressing the "LissajousPlotButton." This function will generate a Lissajous figure from the raw Uapp and raw Q, applying specific colors on the "app.LissajousUIAxes." Additionally, the power value will be displayed on the panel.

```
% Button pushed function: LissajousPlotButton
plot(app.LissajousUIAxes,app.uapp_y,app.q_y,Color='#6a0d83');

% Assign output value of power function to object Power
% which means update power to display
app.Power.Value=power_calculation(app);
```

3.2.3 Callback function: SmoothLissajousSwitchValueChanged

The callback function will activate when the user toggles the "SmoothLissajousSwitch" from "Off" to "On" or vice versa. If the "SmoothLissajousSwitch" is set to "On," this function will generate a Lissajous figure from the filtered Uapp and filtered Q. Otherwise, it will display the original Lissajous figure. The code references "locs_Maxvolt" and "locs_Maxcharge," which are properties, as the smoothed Lissajous figure is designed to plot only the first cycle of the signal.

```
% Value changed function: SmoothLissajousSwitch
% Read the value from checkbox and create if-else condition for
% smoothed/non-smoothed lissajous displays
value = app.SmoothLissajousSwitch.Value;
if(value=="On")
   %% Plot %%

plot(app.LissajousUIAxes,app.uapp_yfilter(app.locs_Maxvolt(1):app.locs_Ma
xvolt(2)),app.q_yfilter(app.locs_Maxvolt(1):app.locs_Maxvolt(2)),Color='#6a0
d83',LineWidth = 1.5);
   ylim(app.LissajousUIAxes,[-2*1e-7,2*1e-7]);
else
   plot(app.LissajousUIAxes,app.uapp_y,app.q_y,Color='#6a0d83');
end
```

3.2.4 Callback function: SmoothSwitchValueChanged

The callback function, specifically "SmoothSwitchValueChanged," will be triggered when the value of "SmoothSwitch" changes. This function is designed to generate a graph of Uapp against time based on the value of "SmoothSwitch." If "SmoothSwitch" is set to "On" and data exists, the graph will depict smoothed Uapp with corresponding configurated time. Conversely, if "SmoothSwitch" is set to "Off" and data exists, the graph will display raw Uapp with the imported time.

```
% Value changed function: SmoothSwitch
value = app.SmoothSwitch.Value;
if(value=="On"&&app.uapp_y(1)~=0)
   % plot graph with smoothed data (Uapp) and time config
elseif(app.uapp_y(1)~=0)
   % plot graph with non-smoothed data (Uapp)
end
```

3.2.5 Callback function: SmoothSwitch_2ValueChanged

The callback function, specifically "SmoothSwitchValueChanged," will be triggered when the value of "SmoothSwitch_2" changes. This function is designed to generate a graph of Q against time based on the value of "SmoothSwitch_2." If "SmoothSwitch" is set to "On" and data exists, the graph will depict smoothed Q with corresponding configurated time. Conversely, if "SmoothSwitch" is set to "Off" and data exists, the graph will display raw Q with the imported time.

```
% Value changed function: SmoothSwitch_2
value = app.SmoothSwitch_2.Value;
if(value=="On"&&app.q_y(1)~=0)
   % plot graph with smoothed data (Q) and time config
elseif(app.q_y(1)~=0)
   % plot graph with non-smoothed data (Q)
end
```

3.2.6 Function: filter_data

The filter_data function is a MATLAB script designed for signal preprocessing, particularly focusing on down sampling and applying a moving average filter. The function takes three input parameters: a placeholder variable (~), an array of time values (xvalue), and an array of signal values (yvalue). The script begins by down sampling the input signal (yvalue) by averaging every 50 consecutive samples, resulting in a down sampled signal (signal_ydown). Subsequently, a moving average filter is applied to further smooth the signal, utilizing a window size of 100 samples. The filtered signal is stored in the array signal_ymov and then assigned to yfilter.

This function is intended to be called in two different callback functions: "ImportButton_UappPushed" callback function and "ImportButton_UcPushed" callback function. Input of the function will be app.uapp_x, app.uapp_y, app.q_x, and app.q_y. Configured time will be assigned to app.t_config. Additionally, the filtered Uapp will be assigned to app.uapp_yfilter and the filtered Q will be assigned to app.q_yfilter.

```
function [yfilter,xfilter] = filter_data(~,xvalue,yvalue)
…
end

% Call in ImportButton_UappPushed, ImportButton_UcPushed %
[app.uapp_yfilter,app.t_config]=filter_data(app,app.uapp_x,app.uapp_y);
[app.q_yfilter,app.t_config]=filter_data(app,app.q_x,app.q_y);
```

```
%% Down Sampling %%

down_size=50;
m=size(yvalue,1);
signal_ydown=zeros(uint64(m/down_size),1);
for i=1:uint64(m/down_size)
   for j=1:down_size
      signal_ydown(i)= signal_ydown(i)+yvalue(down_size*(i-1)+j);
   end
   signal_ydown(i)= signal_ydown(i)/down_size;
end
```

```
%% Moving Average %%

m=size(signal_ydown,1);
moving_size=100; %round(weight);
signal_ymov=zeros(m-moving_size,1);
for i=1:m-moving_size
   for j=1:moving_size
      signal_ymov(i)= signal_ymov(i)+signal_ydown(i+j);
   end
   signal_ymov(i)= signal_ymov(i)/moving_size;
end
```

The function also configures the time axis (xfilter) based on the applied down sampling to the original time values (xvalue). The down sampling factor (down_size) is set to 50. Therefore, the number of elements in the xfilter array would be determined by down sampling the original number of samples in xvalue by a factor of 50. If the original number of samples (size(xvalue, 1)) is 200,000, so the "signal_xdown" array would have 4,000 elements. Plus, the "moving_size" is 100, then the "signal_xmov" array will be 3,900 elements. Significantly, the xfilter array will store the original time values at indices 1,1+50×n, where n iterates until reaching a value of 3,900.

```
%% X-axis /Time config %%

yfilter =signal_ymov;
signal_xdown=xvalue(1:down_size:size(xvalue,1));
signal_xmov=signal_xdown(1:size(signal_xdown,1)-moving_size);
xfilter=signal_xmov(1:size(signal_xmov,1));
```

3.2.7 Function: find_details

The function named “find_details” will extract various details from a given signal (xvalue, yvalue). This function is intended to be called in two different callback functions: “ImportButton_UappPushed” callback function and “ImportButton_UcPushed” callback function. Input of the function will be app.t_config, app.uapp_yfilter, and app.q_yfilter. These input parameters denote the computed characteristics derived from filtered signals, specifically filtered Uapp and Q signals.

The function begins by identifying the lowest and highest points in the input signal (yvalue) using the MATLAB functions min and max, respectively. The root mean square (RMS) value of the signal is computed using the MATLAB function rms.

```
function [min_value, max_value, p2p_value, rms_value, peak, invpeak, f_value] =
find_details (~,xvalue,yvalue)

…

end


% Call in ImportButton_UappPushed, ImportButton_UcPushed %

[app.Min_Uapp.Value, app.Max_Uapp.Value, app.Peaktopeak_uapp.Value,
app.RMS_uapp.Value, app.locs_Maxvolt, app.locs_Minvolt,
app.Freq_uapp.Value] = find_details (app,app.t_config,app.uapp_yfilter);

[app.Min_Q.Value, app.Max_Q.Value, app.Peaktopeak_q.Value,
app.RMS_q.Value, app.locs_Maxcharge, app.locs_Mincharge, app.Freq_q.Value]
= find_details (app,app.t_config,app.q_yfilter);
```

```
%% Find the lowest and highest points %%

% a is the lowest point
% b is the highest point

min_value=min(yvalue);
max_value=max(yvalue);

%% Find RMS %%

% Use RMS matlab function
rms_value=rms(yvalue);
```

The function uses the “findpeaks” function to identify peaks and inverse peaks in the signal. Peaks are detected in the original signal, while inverse peaks are detected in the negated signal (-1 .* yvalue). The minimum peak height is set to the RMS value, and the minimum peak distance is set to 100 samples.

The number of identified peaks is used to iterate through each cycle, calculating the average frequency and peak-to-peak value. The frequency is determined by the reciprocal of the time difference between consecutive peaks, and the P2P value is the average difference between the peak and inverse peak values.

```
[~,peak] = findpeaks(yvalue, 'MinPeakHeight', rms(yvalue), 'MinPeakDistance',
100);
[~,invpeak] = findpeaks(-1.*yvalue ,'MinPeakHeight' ,rms(yvalue),
'MinPeakDistance', 100);

%find the number of peaks
n=size(peak,1);
f=0;
p2p_u=0;

% Use average frequency from every cycles
for i=1:n-1
   f=f+1/(xvalue(peak(i+1))-xvalue(peak(i)));
   p2p_u=p2p_u+yvalue(peak(i))-yvalue(invpeak(i));
end

% Use average p2p and frequency of every peaks
f_value=f/(n-1);
p2p_value= p2p_u/(n-1);
```

3.2.8 Function: power_calculation

The function “power_calculation” performs power calculations for each charging cycle and subsequently computes the average power across all cycles. The 'line' array represents the values of “uapp_yfilter” and “q_yfilter” for one specific charging cycle. The “polyarea” function is then applied to the “line” array, representing a polygon defined by the “uapp_yfilter” and “q_yfilter” values for that cycle. The resulting area is multiplied by 35,000. The computation of “polyarea” is based on the Gauss polygon area formula and the polygon automatically be closed. Because of this reason, when calculate periodic signals, it is necessary to calculate separate cycles.

```
function power_value= power_calculation(app)
…
end

app.Power.Value=power_calculation(app);
```

```
%% Power Calculation
n=size(app.locs_Maxcharge,1);
power=0;

% Calculate the power of each cycle
% We detect n peak so we get n-1 cycle
for i=1:n-1
   locstart=app.locs_Maxcharge(i);
   locstop=app.locs_Maxcharge(i+1);

   m=locstop-locstart+1;
   line=zeros(m,2);
   k=1;
   for j=locstart:locstop
      line(k,1)=app.uapp_yfilter(j);
      line(k,2)=app.q_yfilter(j);
      k=k+1;
   end

   power = power + polyarea(line(:,1),line(:,2))*35000;

% Average power from every cycles
n=size(app.locs_Maxcharge,1);
power_value=power/(n-1);
```

3.3 Program Testing

As you can see from Figure 3., after pressing the “Import” button. A dialogue box will pop up. Then the user can select the file. If the user closes the dialogue box, no graph will be displayed, and a warning message ("User selected Cancel") will be shown via the command window. If the user follows the same order as outlined in the user manual, the graphs and parameters will be displayed. Figure 4 illustrates another option available to the user. Option 2 software was developed using a Savitzky-Golay filter, resulting in parameters that closely match, except for the discharge power, which exhibits more error.

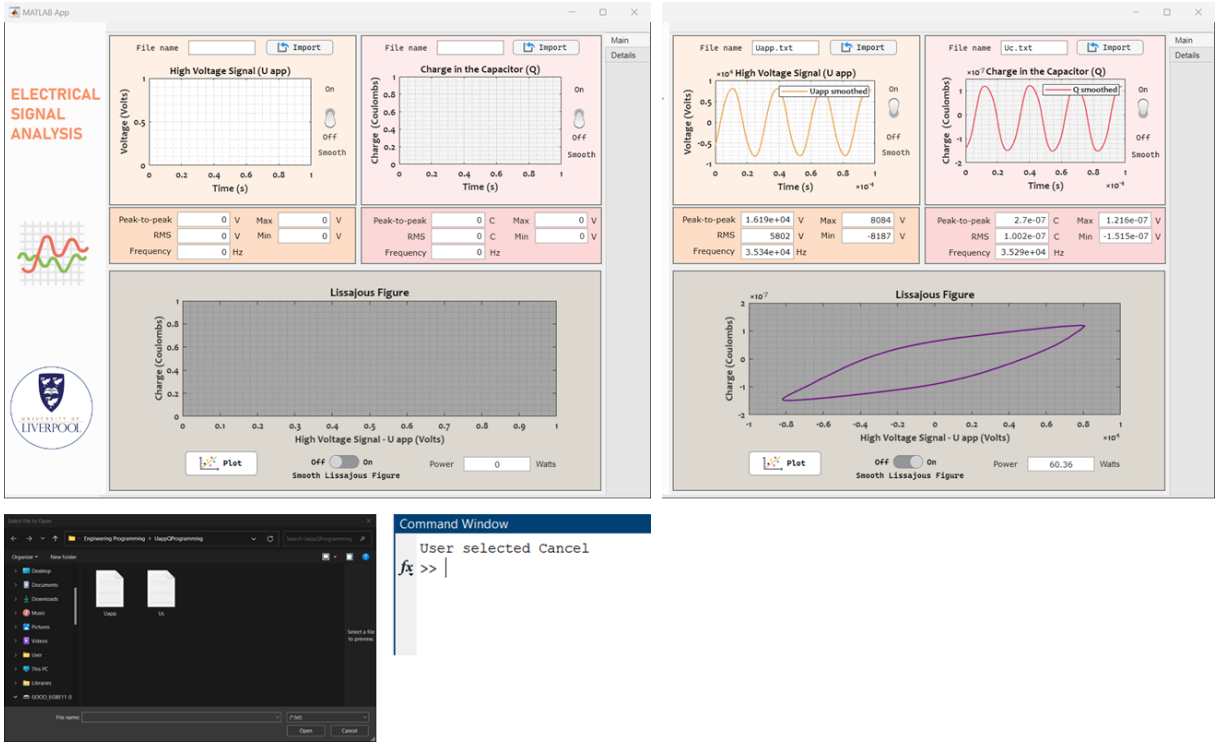

Figure 3: Program testing.

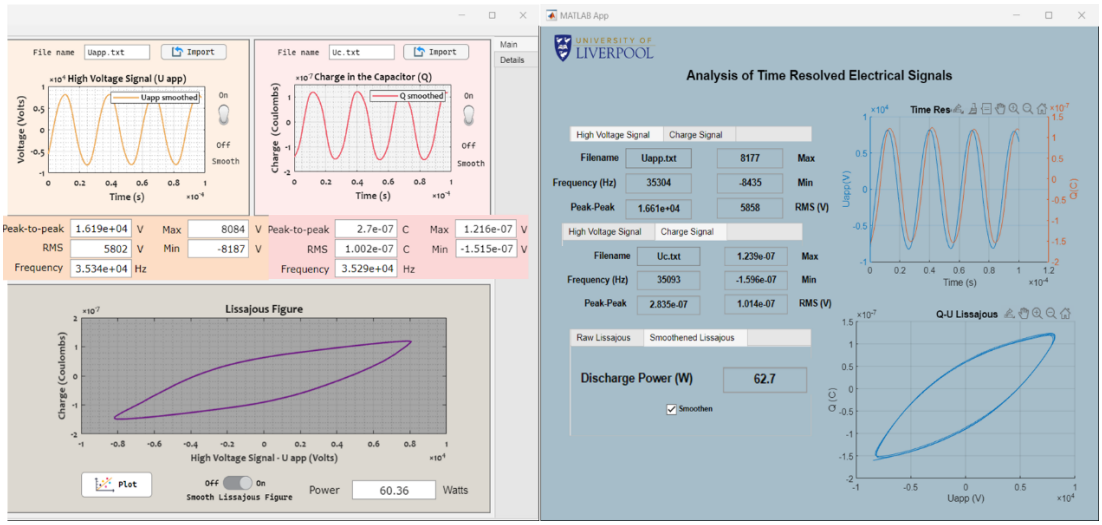

Figure 4: Software: option 1 and option 2.

4. User Manual

Before running the software, ensure that the user places the text files in the same folder as the software. The process to load and display the input file of electrical signals is as follows:

1) Click the “Import” button.
2) A dialogue box will pop up; the user then selects the file. The electrical signal and parameters will appear automatically after successful loading.
3) Click the 'Import' button to load another file.
4) The user selects the file.
5) To display the Lissajous figure and discharge power, click “Plot.”

Note: The order of the 'Import' buttons doesn't matter; the user can select either button first as shown in Figure 5.

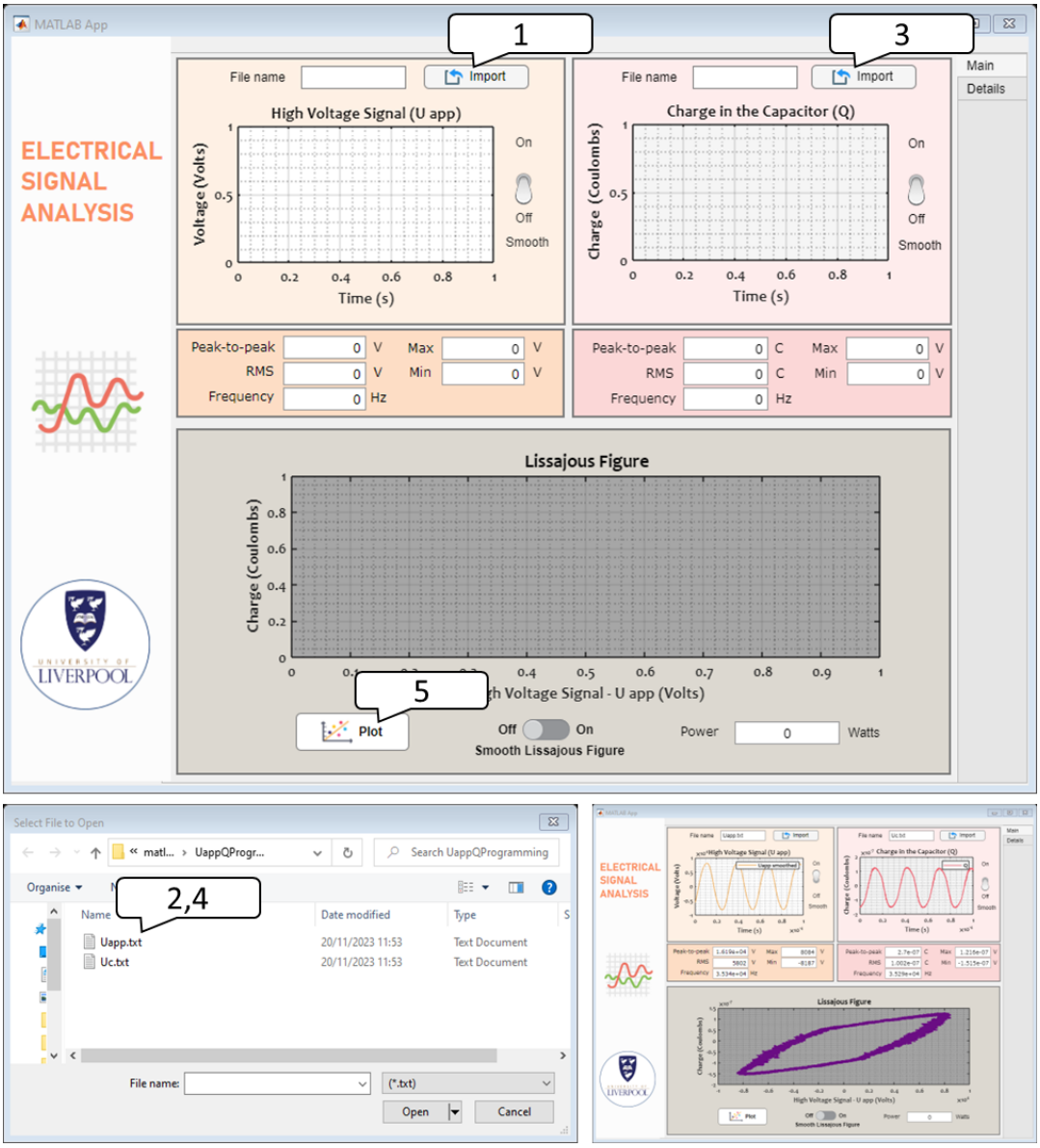

Figure 5: Main panel.

The user also has the options to smooth each graph by pressing the 'Smooth' button, as shown in Figure 6.

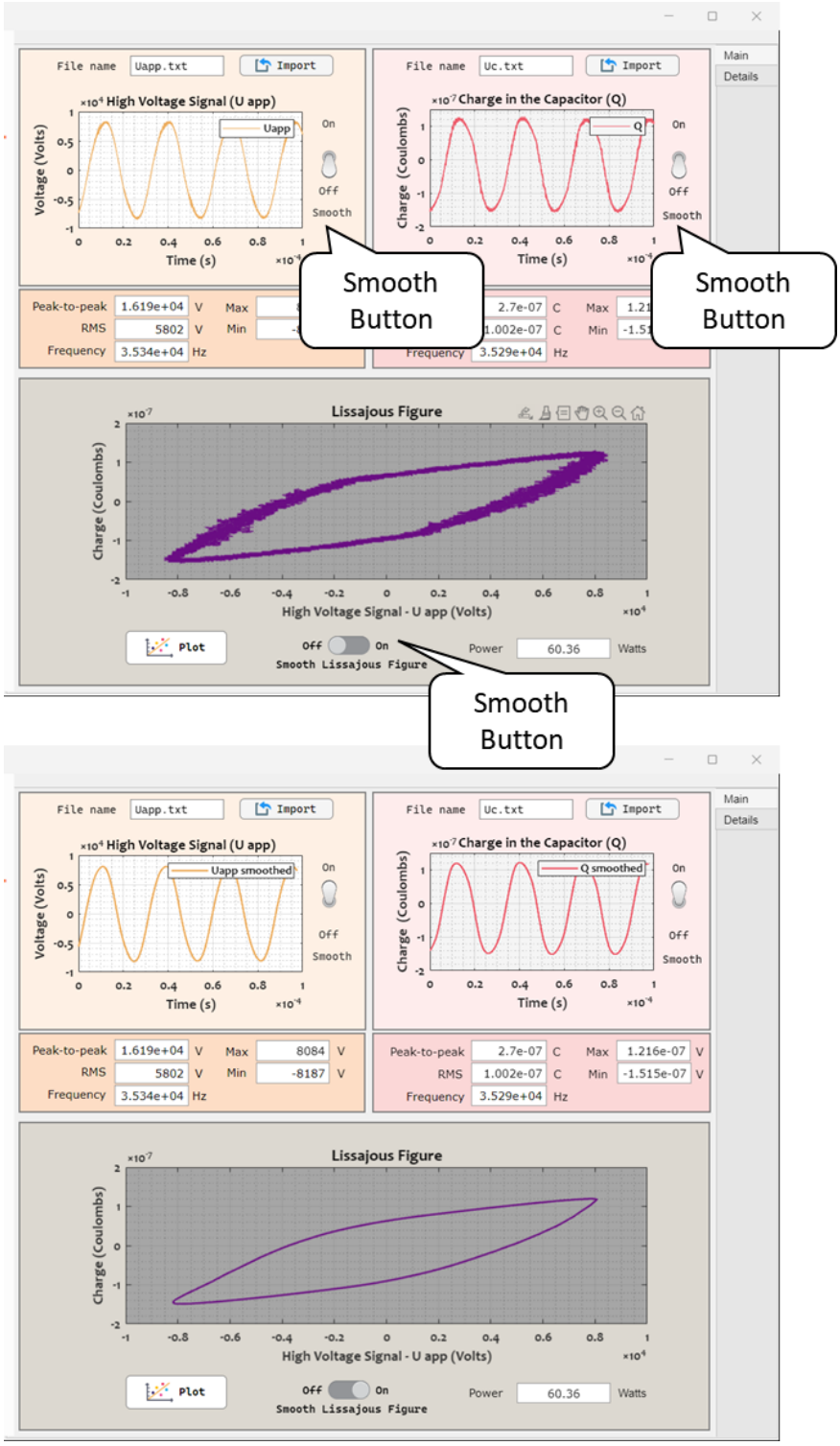

Figure 6: GUI Display - Non-smoothed graphs and smoothed graphs.

If the user clicks on the 'Details' tab, it will display a detailed panel, as shown in Figure 7.

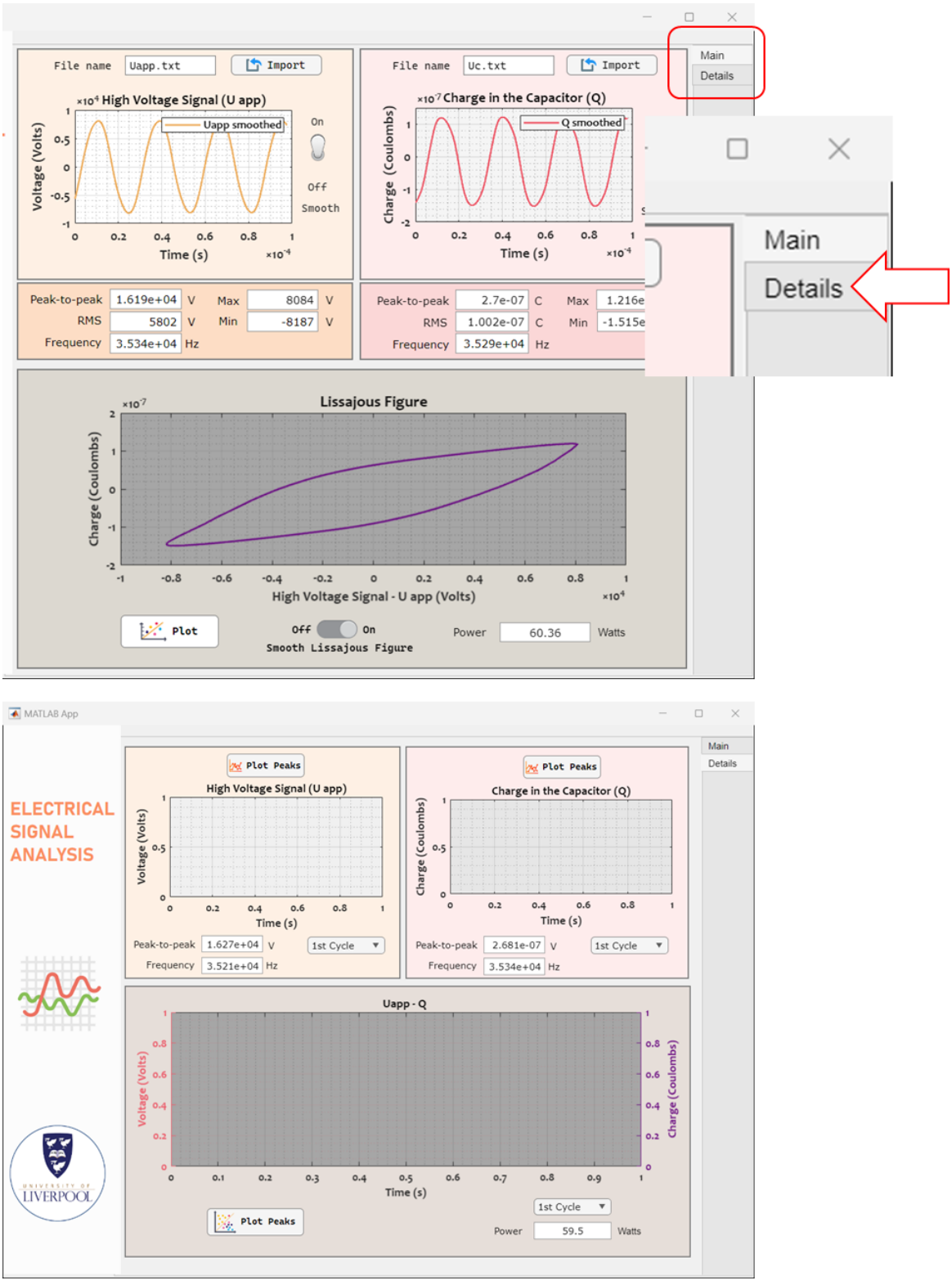

Figure 7: GUI Display – Main panel and details panel.

On the 'Details' panel, after pressing the 'Plot Peaks' button, it will display a signal highlighting peaks and inverse peaks of every cycle. Additionally, there is a feature to present peak-to-peak value, frequency, and power of any cycle by changing the dropdown of each graph as shown in Figure 8.

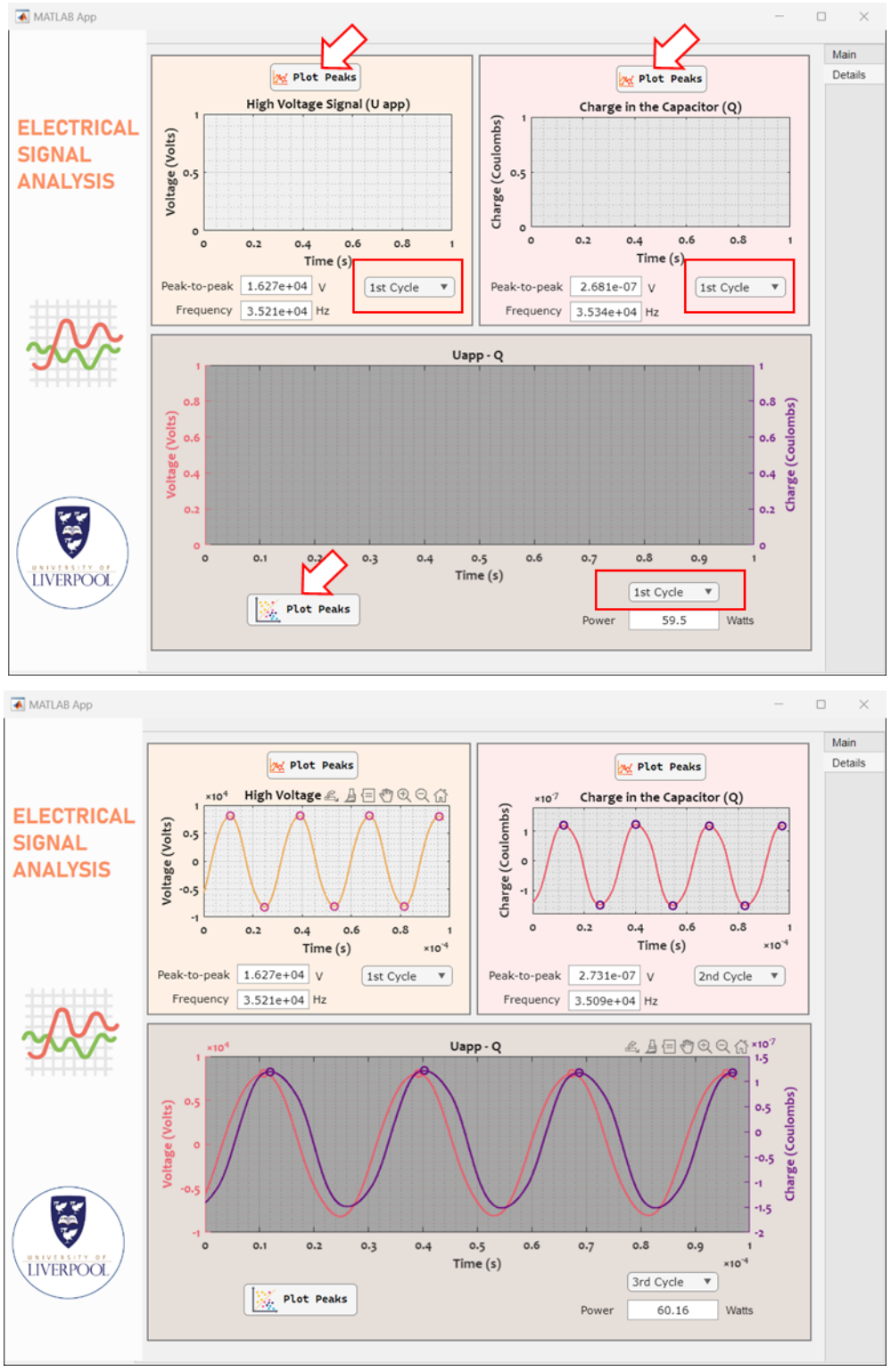

Figure 8: Details tab panel.

5. Discussion and Conclusion

The GUI-based program developed for electrical signal processing stands out for its user-centric design and numerous advantages. The graphical interface not only simplifies data import, visualization, and analysis but also ensures accessibility for users without programming expertise. Real-time interpretation is facilitated by visual feedback, and the option to choose data smoothing provides flexibility in analysis. In the event of varying column counts in input data files, the code intelligently adapts by dynamically increasing variables and employing a for-loop for data extraction. This design choice minimizes the need for code modifications during file imports. The use of "hold on" in graph display allows seamless representation of multiple datasets in the same axes.

In conclusion, the assignment successfully demonstrates the implementation of an efficient and user-friendly GUI-based program for electrical signal processing. The software not only excels in providing immediate visual feedback and streamlined execution of tasks but also accommodates variations in input data seamlessly. The flexibility to analyse specific parameters and smooth data contributes to an enhanced user experience. Overall, the GUI-based program emerges as an accessible, efficient, and user-centric solution for effective signal processing.

Appendix A: MATLAB Code

```
        properties (Access = public)
            uapp_x
            uapp_y
            q_x
            q_y
            t_config %time after config
            uapp_yfilter % uapp after filter
            q_yfilter % q after filter
            locs_Maxvolt %peak position of uapp
            locs_Maxcharge %peak position of q
            locs_Minvolt
            locs_Mincharge
        end
```

```
            % Create find_details function with return values which are lowest
            % and highest points, P2P, RMS, Peak Position, InvPeakPosition,
            % frequency
            function
[min_value,max_value,p2p_value,rms_value,peak,invpeak,f_value]=find_details(~,xvalue
,yvalue)
                %% Find the lowest and highest points %%

                % a is the lowest point
                % b is the highest point

                min_value=min(yvalue);
                max_value=max(yvalue);

                %% Find RMS %%

                % Use RMS matlab function
                rms_value=rms(yvalue);
```

```
        %% Find Every Peaks (Calculate Frequency & P2P)

        % Use findpeaks to find the peaks (refers to maximum points) of signal
        % Set the distance equals 100 means that the next peak should be away
from current peak more than 100 samples
        % the samples after filtered is 3900 samples, so if the input signal change,
this number should clarify again
        % for complex signals, more parameters should add as the arguments of
findpeaks function


        % peak refers to the peak of the graph while invpeak refers to pposite of
the peak of the graph
        % p2p can find from the highest minus the lowest point of each cycle

        % Set the reference point equals RMS of their signal to make sure that
Peak point is not a spike from noise

        [~,peak] =
findpeaks(yvalue,'MinPeakHeight',rms(yvalue),'MinPeakDistance',100);
        [~,invpeak] = findpeaks(-
1.*yvalue,'MinPeakHeight',rms(yvalue),'MinPeakDistance',100);

        %find the number of peaks
        n=size(peak,1);
        f=0;
        p2p_u=0;

        % Use average frequency from every cycles
        for i=1:n-1
           f=f+1/(xvalue(peak(i+1))-xvalue(peak(i)));
           p2p_u=p2p_u+yvalue(peak(i))-yvalue(invpeak(i));
        end

        % Use average p2p and freq of every peaks
        f_value=f/(n-1);
        p2p_value= p2p_u/(n-1);

        % Use only first cycle to calculate frequency
        % f1=1/(app.t_config(locs_Maxvolt(2))-app.t_config(locs_Maxvolt(1)));

    end
```

```
function [yfilter,xfilter] = filter_data(~,xvalue,yvalue)
        %% Down Sampling %%
        % Set sampling size = 50
        % so the samples from 200000 will be 4000 samples
        % Average 50 samples and keep value in a sample
        down_size=50;
        m=size(yvalue,1);
        signal_ydown=zeros(uint64(m/down_size),1);
        for i=1:uint64(m/down_size)
           for j=1:down_size
              signal_ydown(i)= signal_ydown(i)+yvalue(down_size*(i-1)+j);
           end
           signal_ydown(i)= signal_ydown(i)/down_size;
        end

        %% Moving Average %%
        % Set the moving average size =100
        % so the samples from 4000 with be 3900 samples
        % this moving average function cut the last 100 samples
        % if the signal is unknown, I would suggest to use zero-padding
        % or nearest border as values for padding

        m=size(signal_ydown,1);
        moving_size=100; %round(weight);
        signal_ymov=zeros(m-moving_size,1);
        for i=1:m-moving_size
           for j=1:moving_size
              signal_ymov(i)= signal_ymov(i)+signal_ydown(i+j);
           end
           signal_ymov(i)= signal_ymov(i)/moving_size;
        end

        %% X-axis /Time config %%
        % Configurate time by the description of the filters that
        % applied to signals
        yfilter =signal_ymov;
        signal_xdown=xvalue(1:down_size:size(xvalue,1));
        signal_xmov=signal_xdown(1:size(signal_xdown,1)-moving_size);
        xfilter=signal_xmov(1:size(signal_xmov,1));

    end
```

```
% create power calculation function with return value
    function power_value= power_calculation(app)
      %% Power Calculation
      n=size(app.locs_Maxcharge,1);
      power=0;

      % Calculate the power of each cycle
      % We detect n peak so we get n-1 cycle
      for i=1:n-1
        locstart=app.locs_Maxcharge(i);
        locstop=app.locs_Maxcharge(i+1);

        m=locstop-locstart+1;
        line=zeros(m,2);
        k=1;
        for j=locstart:locstop
          % Create the "line" array (size=k*2) representing value
          % of Uapp and Q in each cycle
          line(k,1)=app.uapp_yfilter(j);
          line(k,2)=app.q_yfilter(j);
          k=k+1;
        end
        % Power of each cycle
        %The computation of “polyarea” is based on the Gauss polygon
        % area formula and the polygon automatically be closed.
        % Because of this reason, when calculate periodic signals,
        % it is necessary to calculate separate cycles.
        power = power + polyarea(line(:,1),line(:,2))*35000;

      end

      % Average power from every cycles
      n=size(app.locs_Maxcharge,1);

      power_value=power/(n-1);
    end
```

```
        function [p2p_u,f] = find_p2pfq(~,xvalue,yvalue,order)
                [~,peak] =
findpeaks(yvalue,'MinPeakHeight',rms(yvalue),'MinPeakDistance',100);
                [~,invpeak] = findpeaks(-
1.*yvalue,'MinPeakHeight',rms(yvalue),'MinPeakDistance',100);

                f=0;
                p2p_u=0;

                % Use average frequency from every cycles
                i=order;
                f=f+1/(xvalue(peak(i+1))-xvalue(peak(i)));
                p2p_u=p2p_u+yvalue(peak(i))-yvalue(invpeak(i));

            end

            function power = power_cycle(app,order)
                %% Power Calculation
                power=0;

                % Calculate the power of each cycle
                % We detect n peak so we get n-1 cycle
                locstart=app.locs_Maxvolt(order);
                locstop=app.locs_Maxvolt(order+1);

                m=locstop-locstart+1;
                line=zeros(m,2);
                k=1;
                for j=locstart:locstop
                    % Create the "line" array (size=k*2) representing value
                    % of Uapp and Q in each cycle
                    line(k,1)=app.uapp_yfilter(j);
                    line(k,2)=app.q_yfilter(j);
                    k=k+1;
                end
                % Power of each cycle
                %The computation of “polyarea” is based on the Gauss polygon
                % area formula and the polygon automatically be closed.
                % Because of this reason, when calculate periodic signals,
                % it is necessary to calculate separate cycles.
                power = power + polyarea(line(:,1),line(:,2))*35000;

            end
```

```
% Code that executes after component creation
    function startupFcn(app)
        clc;
        app.Peaktopeak_q.Value=0;
        app.Peaktopeak_uapp.Value=0;
        app.RMS_q.Value=0;
        app.RMS_uapp.Value=0;
        app.Freq_q.Value=0;
        app.Freq_uapp.Value=0;
        app.Min_Uapp.Value=0;
        app.Max_Uapp.Value=0;
        app.Min_Q.Value=0;
        app.Max_Q.Value=0;
        app.Power.Value=0;
        app.uapp_x(1)=0;
        app.uapp_y(1)=0;
        app.q_x(1)=0;
        app.q_y(1)=0;

        % Set Peak Display of Uapp and Q Axis
        yyaxis(app.UappQPeaksAxes,"left")
        ylabel(app.UappQPeaksAxes,'Voltage (Volts)')
        yyaxis(app.UappQPeaksAxes,"right")
        ylabel(app.UappQPeaksAxes,'Charge (Coulombs)')
        axis(app.UappQPeaksAxes,"auto")

    end
```

```
        % Button pushed function: ImportButton_Uapp
            function ImportButton_UappPushed(app, event)
                [filename,]=uigetfile('*.txt');

                % Create If-else condition for the situation that user click
                % cancel preventing an error that the app doesn't get any
                % input table from selected file

                if isequal(filename,0)
                    disp('User selected Cancel');

                else
                    %% importdata %%
                    app.FilenameUapp.Value=filename;
                    T=readtable(filename);

                    app.uapp_x = table2array(T(:,1));
                    app.uapp_y = table2array(T(:,2));
                    plot(app.UappUIAxes,app.uapp_x,app.uapp_y,Color='#eeaf61');
                    legend(app.UappUIAxes,"Uapp");
                    xlim(app.UappUIAxes,[0 1e-4]);
                    ylim(app.UappUIAxes,"auto");

                    %%
                    % Filter function %
                    % input is uapp_x and uapp_y
                    % output is uapp_yfilter and t_config

[app.uapp_yfilter,app.t_config]=filter_data(app,app.uapp_x,app.uapp_y);

                    % Find details function %

                    % update min, max, p2p, rms, locus_max, locus_min, and freq
                    % of Uapp to receive output from function find details

[app.Min_Uapp.Value,app.Max_Uapp.Value,app.Peaktopeak_uapp.Value,app.RMS_uapp
.Value,app.locs_Maxvolt,app.locs_Minvolt,app.Freq_uapp.Value]=find_details(app,app.t
_config,app.uapp_yfilter);

                end

            end
```

```
        % Button pushed function: ImportButton_Uc
            function ImportButton_UcPushed(app, event)
                % The function works as same as Uapp callback function, but
                % change the object to display

                [filename, ]=uigetfile('*.txt');

                if isequal(filename,0)
                    disp('User selected Cancel');

                else
                    %% importdata %%
                    app.FilenameUc.Value=filename;
                    T=readtable(filename);
                    app.q_x = table2array(T(:,1));
                    app.q_y = 22*1e-9.*table2array(T(:,2));
                    plot(app.QUIAxes,app.q_x,app.q_y,Color='#ee5d6c');
                    legend(app.QUIAxes,"Q");
                    xlim(app.QUIAxes,[0 1e-4]);
                    ylim(app.QUIAxes,"auto");
                    %%
                    % Filter function %
                    [app.q_yfilter,app.t_config]=filter_data(app,app.q_x,app.q_y);
                    % Find details function %

[app.Min_Q.Value,app.Max_Q.Value,app.Peaktopeak_q.Value,app.RMS_q.Value,app.locs_Maxcharge,app.locs_Mincharge,app.Freq_q.Value]=find_details(app,app.t_config,app.q_yfilter);

                end
            end
```

```
        % Button pushed function: LissajousPlotButton
            function LissajousPlotButtonPushed(app, event)
                plot(app.LissajousUIAxes,app.uapp_y,app.q_y,Color='#6a0d83');

                % Assign output value of power function to object Power
                % which means update power to display
                app.Power.Value=power_calculation(app);
            end

            % Value changed function: SmoothSwitch
            function SmoothSwitchValueChanged(app, event)
                %Uapp
                value = app.SmoothSwitch.Value;
                if(value=="On"&&app.uapp_y(1)~=0)
                    % plot graph with smoothed data and time config

plot(app.UappUIAxes,app.t_config,app.uapp_yfilter,Color='#eeaf61',LineWidth = 1.5);
                    legend(app.UappUIAxes,'Uapp smoothed');
                elseif(app.uapp_y(1)~=0)
                    % plot graph with non-smoothed data
                    plot(app.UappUIAxes,app.uapp_x,app.uapp_y,Color='#eeaf61');
                    legend(app.UappUIAxes,"Uapp");
                end
            end

            % Value changed function: SmoothSwitch_2
            function SmoothSwitch_2ValueChanged(app, event)
                %Q
                value = app.SmoothSwitch_2.Value;
                if(value=="On"&&app.q_y(1)~=0)
                    % plot graph with smoothed data and time config

plot(app.QUIAxes,app.t_config,app.q_yfilter,Color='#ee5d6c',LineWidth = 1.5);
                    legend(app.QUIAxes,'Q smoothed');
                elseif(app.q_y(1)~=0)
                    % plot graph with non-smoothed data
                    plot(app.QUIAxes,app.q_x,app.q_y,Color='#ee5d6c');
                    legend(app.QUIAxes,"Q");
                end
            end
```

```
        % Value changed function: SmoothLissajousSwitch
            function SmoothLissajousSwitchValueChanged(app, event)
                % Read the value from checkbox and create if-else condition for
                % smoothed/non-smoothed lissajous displays
                value = app.SmoothLissajousSwitch.Value;
                if(value=="On")
                    %% Plot %%

plot(app.LissajousUIAxes,app.uapp_yfilter(app.locs_Maxvolt(1):app.locs_Maxvolt(2)),ap
p.q_yfilter(app.locs_Maxvolt(1):app.locs_Maxvolt(2)),Color='#6a0d83',LineWidth = 1.5);
                    ylim(app.LissajousUIAxes,[-2*1e-7,2*1e-7]);
                else
                    plot(app.LissajousUIAxes,app.uapp_y,app.q_y,Color='#6a0d83');
                end
            end

            % Button pushed function: PlotPeaksButton
            function PlotPeaksButtonPushed(app, event)
                [pk1,lk1] =
findpeaks(app.uapp_yfilter,'MinPeakHeight',app.RMS_uapp.Value,'MinPeakDistance',10
0);
                [pk2,lk2] =
findpeaks(app.q_yfilter,'MinPeakHeight',app.RMS_q.Value,'MinPeakDistance',100);

                yyaxis(app.UappQPeaksAxes,"left")

plot(app.UappQPeaksAxes,app.t_config,app.uapp_yfilter,app.t_config(lk1),pk1,'o',LineW
idth = 1.5)
                ylabel(app.UappQPeaksAxes,'Voltage (Volts)')

                yyaxis(app.UappQPeaksAxes,"right")

plot(app.UappQPeaksAxes,app.t_config,app.q_yfilter,app.t_config(lk2),pk2,'o',LineWidth
= 1.5)
                ylabel(app.UappQPeaksAxes,'Charge (Coulombs)')
                axis(app.UappQPeaksAxes,"auto")

            end
```

```
        % Button pushed function: PlotPeaksButton_2
            function PlotPeaksButtonUPushed(app, event)
                [pk1,lk1] =
findpeaks(app.uapp_yfilter,'MinPeakHeight',app.RMS_uapp.Value,'MinPeakDistance',10
0);
                [pk2,lk2] = findpeaks(-
1*app.uapp_yfilter,'MinPeakHeight',app.RMS_uapp.Value,'MinPeakDistance',100);


plot(app.UappPeaksAxes,app.t_config,app.uapp_yfilter,Color='#eeaf61',LineWidth = 1.5)
                hold(app.UappPeaksAxes, 'on' )

plot(app.UappPeaksAxes,app.t_config(lk1),pk1,'o',Color='#ce4993',LineWidth = 1.5)

                plot(app.UappPeaksAxes,app.t_config(lk2),-
pk2,'o',Color='#ce4993',LineWidth = 1.5)
                axis(app.UappPeaksAxes,"auto")


            end

            % Button pushed function: PlotPeaksButton_3
            function PlotPeaksButtonQPushed(app, event)
                [pk1,lk1] =
findpeaks(app.q_yfilter,'MinPeakHeight',app.RMS_q.Value,'MinPeakDistance',100);
                [pk2,lk2] = findpeaks(-
1*app.q_yfilter,'MinPeakHeight',app.RMS_q.Value,'MinPeakDistance',100);


plot(app.QPeaksAxes,app.t_config,app.q_yfilter,Color='#ee5d6c',LineWidth = 1.5)
                hold(app.QPeaksAxes, 'on' )
                plot(app.QPeaksAxes,app.t_config(lk1),pk1,'o',Color='#6a0d83',LineWidth
= 1.5)
                plot(app.QPeaksAxes,app.t_config(lk2),-
pk2,'o',Color='#6a0d83',LineWidth = 1.5)
                axis(app.QPeaksAxes,"auto")
                ylim(app.QPeaksAxes,[-1.8*1e-7,1.5*1e-7]);

            end
```

```
        % Button down function: DetailsTab
            function DetailsTabButtonDown(app, event)
                % when user click on details tab, the startup value of p2p,
                % freq and power will automatically calculate from first cycle
                % until the dropdown value change
                app.Power_2.Value=power_cycle(app,1);

[app.Peaktopeak_uapp_2.Value,app.Freq_uapp_2.Value]=find_p2pfq(app,app.t_config,ap
p.uapp_yfilter,1);

[app.Peaktopeak_q_2.Value,app.Freq_q_2.Value]=find_p2pfq(app,app.t_config,app.q_yfi
lter,1);
            end

            % Value changed function: DropDown
            function DropDownValueChanged(app, event)
                % when the dropdown value change, switch case was used to
                % assign value of order
                value = app.DropDown.Value;
                switch value
                    case '1st Cycle'
                        order=1;
                    case '2nd Cycle'
                        order=2;
                    case '3rd Cycle'
                        order=3;
                    otherwise
                        order=1;
                end
                % Find details function %
                % In this function only p2p and freq will update and order will
                % be input parameter tell what cycle data user would like to
                % know

[app.Peaktopeak_uapp_2.Value,app.Freq_uapp_2.Value]=find_p2pfq(app,app.t_config,ap
p.uapp_yfilter,order);

            end
```

```
        % Value changed function: DropDown_2
            function DropDown_2ValueChanged(app, event)
                value = app.DropDown_2.Value;
                switch value
                    case '1st Cycle'
                        order=1;
                    case '2nd Cycle'
                        order=2;
                    case '3rd Cycle'
                        order=3;
                    otherwise
                        order=1;
                end

[app.Peaktopeak_q_2.Value,app.Freq_q_2.Value]=find_p2pfq(app,app.t_config,app.q_yfi
lter,order);
            end

            % Value changed function: DropDown_3
            function DropDown_3ValueChanged(app, event)
                value = app.DropDown_3.Value;
                switch value
                    case '1st Cycle'
                        order=1;
                    case '2nd Cycle'
                        order=2;
                    case '3rd Cycle'
                        order=3;
                    otherwise
                        order=1;
                end
                app.Power_2.Value=power_cycle(app,order);

            end
```