Experiment 1: Programming and Data Analysis in LabWindows/CVI

Elise Jortberg
Lab Partner: Dena Guo
Jan 28, 2016

1. INTRODUCTION

The primary purpose of this exercise is to create a program that automates analysis of a data set. This lab introduced the principles of data analysis with a programming language. Using LabWindows, we designed a program to import data, remove the baseline, normalize, calculate the centroid, and output the result. The program consists of two parts: a GUI (graphical user interface) and corresponding C script to execute the commands.

The simple GUI allows the user to easily interact with the data and perform the necessary calculations without having to hard-code parameters. LabWindows utilizes the efficiency of C, but also contains a set of proprietary libraries. These built in functions construct the terminal for basic functionality between GUI and C script, such as inputting and outputting data, and displaying numerics and graphs. The C script contains the callback functions, which perform the desired commands (i.e. input file). In our case, the analysis required three main calculation steps: removing the baseline, normalization, and locating the centroid.

2. MATERIALS AND METHODS

We used National Instrument's Lab Windows CVI to build the program. According to National Instruments, LabWindows is "an ANSI C integrated development environment and engineering toolbox with built-in libraries for measurement, analysis, and engineering UI design." Each element of the GUI, shown in Figure 1, has an associated callback function. The essential functions are input/output, remove baseline, normalize data, and calculate centroid.

2a) Input and Output

The select file button prompts the user to select a file to analyze. The analyzed data used for this lab is an 131x2 array of (x,y) values. The inputted file, "1.txt" is a tab delimited text file with the listed indices (x) followed by the f(x) values (y). The LabWindows function FileToArray() saves the inputted data to one large 1 dimensional array. We then determine how many elements are present and separate the data into separate x and y arrays:

```
// Determine Number of Total Elements in Input File
      for (i=0; i<1000; i++){
                if (InputArray[i])
                { cntr++;}
                else
                {continue;}
               }
           newarraysize = cntr/2;
```

Xarray and Yarray are initialized to contain ‘newarraysize’ elements. New array size equals the length of our input file, 131. We then iterate through our Input Array ‘cntr’

times. 'Cntr' is 232, the total number of values x and y. The values are assigned to x if the value is even, y if odd.

```
// Split InputArray into X- and Y- Arrays
          double xarray[newarraysize];
          double yarray[newarraysize];

          int ycntr =0;
          int xcntr =0;
          for (i=0; i<cntr; i++){
              if (i % 2){
                 y[ycntr]= InputArray[i];
                 ycntr++; }
              else{
                  x[xcntr] = InputArray[i];
                  xcntr++;
              }
               }
```

The new arrays are stored as global variables, so they can be called from other functions for further analysis.

Just as we read in the file, we follow a similar process to write the data to a new output file. When the user clicks the Save command, the callback function Save_Normalized_Data organizes the x and y data into one long array listing all x then all y values. The LabWindows function ArrayToFile() saves this data into a user specified .txt file, formatted into two columns by (x,y) value.

```
ArrayToFile (filename, outputarray, VAL_DOUBLE, (2*newarraysize), 2,
             VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
             VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_APPEND);
```

This output file is now accessible by other programs, i.e. Excel, for further analysis or sharing.

2b) Removing the Baseline

To remove the constant background from the data, the user needs to subtract the baseline from the y array. The user enters the value to be subtracted into the 'Estimate Baseline' input and selects 'Remove Baseline.' The callback function loops through the y values and subtracts the baseline value from each element in the array:

```
GetCtrlVal (panelHandle, PANEL_NUMERIC_BL, &bl);
            for (i=0; i<newarraysize; i++)
            {
             yprime[i] = y[i] - bl;   //remove baseline, bl=inputted val
            }
```

The user has the option to compare the updated data to the original on the same graph (see Figure 1) by hitting 'Update Graph,' which performs:

```
      // Plot Graph (Input Data)
            PlotXY (panelHandle, PANEL_GRAPH, x, y,
                newarraysize, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE,
                VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_RED);
```

2c) Normalizing the Data

Normalizing the data normalizes the units of the data to an arbitrary scale. 'Normalizing' could be one many different processes, i.e. removing mean then dividing by standard deviation or setting maximum to 1. For this data we normalize by multiplying the data by normalization constant, norm_c. Norm_c is the area under the curve per length of

array. We calculated the area by taking the cumulative sum of the y values and then norm_c by dividing this area by the length of y array (equal to newarraysize). Multiplying each element by norm_c effectively sets the measured area under the total curve equal to 1.

```
for (i=0; i<newarraysize; i++){
          norm_c= norm_c + yprime[i];     //calculate area
       }
       area = norm_c;
       // Normalize Curve
       norm_c = norm_c/newarraysize;      //normalization constant

       for (i=0; i<newarraysize; i++){
           yprime_norm[i] = norm_c*yprime[i];   //normalize
}
```

Upon calculation, the Normalize() function automatically displays the calculated normalization constant and updates the bottom graph with the normalized data (Figure 1).

```
SetCtrlVal (panelHandle, PANEL_NUMERIC_NORM_C, norm_c);
// Plot Normalized Curve
PlotXY (panelHandle, PANEL_GRAPH_2, x, yprime_norm, newarraysize,
    VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
    VAL_SOLID, 1, VAL_GREEN);
```

2d) Calculating the Centroid

The centroid of the data is the point where f(x) is equal to half of the total area. Using the normalized data, we loop through y values and calculate the cumulative area. The centroid is the point at which this area equals 0.5.

```
for(i=0; i<newarraysize; i++){
    area_norm= area_norm + yprime_norm[i];
    if (area_norm >= (0.5))  // area under a normalized curve = 1
        {
             centroid = i;
             SetCtrlVal (panelHandle, PANEL_NUMERIC_CNTD, centroid);
             break;
        }
```

Once the program identifies the centroid, the loop breaks and the GUI displays the centroid value.

3. RESULTS

Figure 1 displays the GUI created by our program. The top graph displays the original data (red), the baseline removed data (blue), and normalized data (green). Once saved, we opened the output file and re-plotted the normalized data in Microsoft Excel (Figure 2).

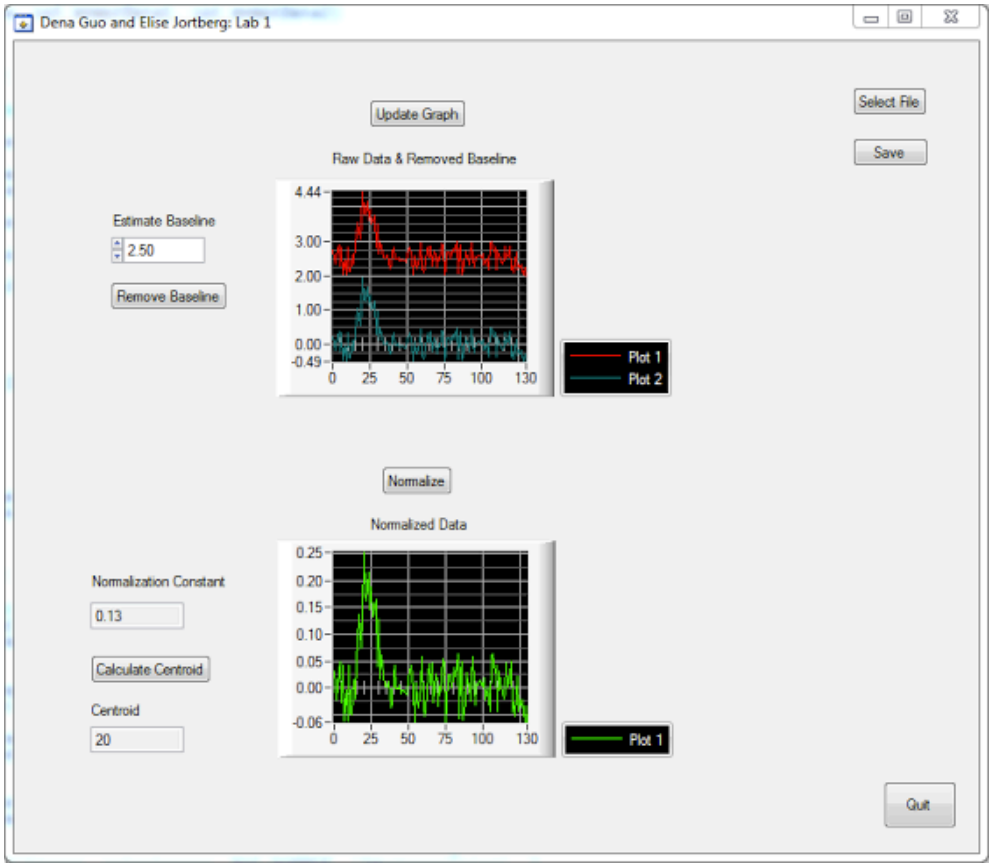

Figure 1: Data Analysis GUI

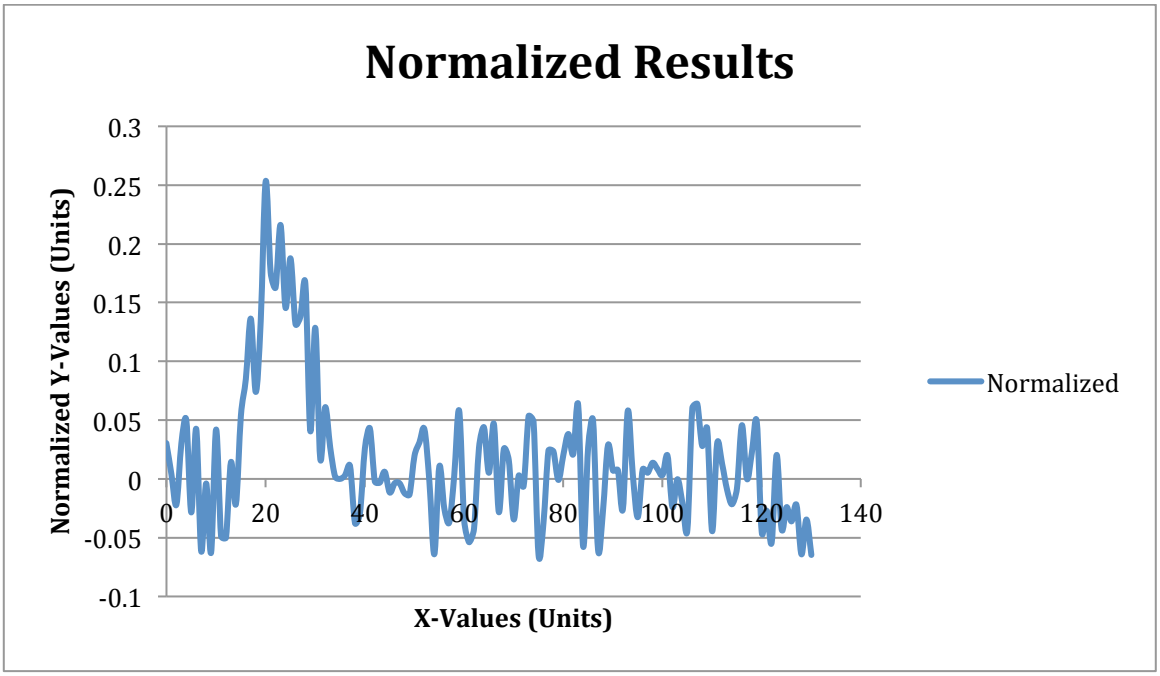

Figure 2: Normalized Data in Excel

4. DISCUSSION

This lab explained how to utilize the functionality already provided by LabWindows and expand on this by programming additional tasks using C. Our program allows the user to quickly read in and normalize data, an important step in most experiments. This configurable program provides the framework for future experimental analysis. For full source code, please see Appendix.

5. CONCLUSIONS

Using LabWindows, we were able to build a program that accurately read in data, removed the baseline, normalized, and calculate the centroid. The GUI displayed and plotted the results of these calculations. The normalized data can be saved in a shareable format, i.e. 'output.txt.' The program successfully analyzed the '1.txt' file, displayed the results, and then saved these results to an external file. This program may now be used to repeat these processes for future experiments.

APPENDIX

Source code is located in the Dena_Elise folder in the Student directory.

```
#include <formatio.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include <stdio.h>

#include "Exp_pt2.h"
//Global Variables
static double InputArray[1000]; //Initialze arrays
static int panelHandle;
static double x[1000];
static double y[1000];
static double yprime[1000];
static double yprime_norm[1000];
int newarraysize;
static double norm_c =0;
static int centroid;
static double area;

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1;  // out of memory
    if ((panelHandle = LoadPanel (0, "Exp_pt2.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    DiscardPanel (panelHandle);
    return 0;
}


int CVICALLBACK Bye (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{ //Quits the program
    switch (event)
    {
```

```
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

int CVICALLBACK InputData (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
         {
            // Select Input File
            FileSelectPopup ("", "*.*", "", "", VAL_LOAD_BUTTON, 0, 0, 1, 0,
FileName);
            FileToArray (FileName, InputArray, VAL_DOUBLE, 1000, 1,
             VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS, VAL_ASCII);
            // Determine Number of Total Elements in Input File
            int cntr = 1;
            int i;

            for (i=0; i<1000; i++){
                if (InputArray[i])
                {
                    cntr++;
                }
                else
                {
                    continue;
                }
               }

            newarraysize = cntr/2;
            // Split InputArray into X- and Y- Arrays
            double xarray[newarraysize];
            double yarray[newarraysize];
            int ycntr =0;
            int xcntr =0;
            for (i=0; i<cntr; i++){
                if (i % 2){
                   yarray[ycntr] = InputArray[i];
                   y[ycntr]= InputArray[i];
                   ycntr++; }
                else{
                    xarray[xcntr] = InputArray[i];
                    x[xcntr] = InputArray[i];
                    xcntr++;
                }
                 }
         }
            break;
        case EVENT_RIGHT_CLICK:
            break;
    }
```

```
    return 0;
}

int CVICALLBACK UpdateGraph (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            // Plot Graph (Input Data)
            PlotXY (panelHandle, PANEL_GRAPH, x, y,
                newarraysize, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE,
                VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_RED);
            break;
        case EVENT_RIGHT_CLICK:
            break;
    }
    return 0;
}

int CVICALLBACK remove_baseline (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            // Remove Baseline from Y Values
            int i;
            double bl; //value to remove from baseline
            GetCtrlVal (panelHandle, PANEL_NUMERIC_BL, &bl);
            for (i=0; i<newarraysize; i++)
            {
             yprime[i] = y[i] - bl;   //remove baseline
            }
            // Plot New Graph
            PlotXY (panelHandle, PANEL_GRAPH, x, yprime,
                newarraysize, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE,
                VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_DK_CYAN); //add baseline
to graph figure
            break;
        case EVENT_RIGHT_CLICK:
            break;
    }
    return 0;
}

int CVICALLBACK Normalize (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            // Calculate Area Under the Curve
            int i;
            double area;

            for (i=0; i<newarraysize; i++){
               norm_c= norm_c + yprime[i];     //calculate area
     }
```

```
            area = norm_c;
            // Normalize Curve
            norm_c = norm_c/newarraysize;      //normalization constant
            for (i=0; i<newarraysize; i++){
                yprime_norm[i] = norm_c*yprime[i];   //normalize by
multiplying each element by norm_c
            }

            SetCtrlVal (panelHandle, PANEL_NUMERIC_NORM_C, norm_c);
            // Plot Normalized Curve
            PlotXY (panelHandle, PANEL_GRAPH_2, x, yprime_norm, newarraysize,
                VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
                VAL_SOLID, 1, VAL_GREEN);
            break;
        case EVENT_RIGHT_CLICK:
            break;
    }
    return 0;
}

int CVICALLBACK Centroid (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
           // Calculate Centroid
           int i;
           double area_norm = 0;

           for(i=0; i<newarraysize; i++){
              area_norm= area_norm + yprime_norm[i];
              if (area_norm >= (0.5))  // area under a normalized curve = 1
              {
                   centroid = i;
                   SetCtrlVal (panelHandle, PANEL_NUMERIC_CNTD, centroid);
                   break;
              }
            }

            break;
        case EVENT_RIGHT_CLICK:
            break;
    }
    return 0;
}

int CVICALLBACK Save_Normalized_Data (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            // Save Output File
            char filename[300];
            double outputarray[1000];
            int i;

            FileSelectPopup ("", "*.*", "", "", VAL_SAVE_BUTTON, 0, 0, 1, 1,
```

```
                filename);

            for(i=0; i< (newarraysize); i++){
               outputarray[i] = x[i];
            }
            for (i = newarraysize; i < (2*newarraysize); i++){
                outputarray[i] = yprime_norm[i - newarraysize];

                                                                    }
            ArrayToFile (filename, outputarray, VAL_DOUBLE, (2*newarraysize),
2,
             VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
             VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_APPEND);
            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}
```