Laboratory 5: Compute the histogram of the array

The aims of today's lab are:

- Finish last lab on Sum of Absolute Error (SAE) and normalised cross-correlation (NCC).
- Compute the histogram of the array used last time (in Lab 4). You'll need to extend the `MyVector` class and add new tests in `test_my_vector.cpp`.

Task 0: Finish last lab

Before starting something new, finish the lab from last time.

Task 1: Compute the histogram

An histogram is a graphical representation of the intensity distribution in a signal. It plots the number of samples for each intensity value. This definition can be adapted to 2D images: An histogram is a graphical representation of the pixel intensity distribution in a digital image. It plots the number of pixel for each intensity value.

The data is grouped into ranges (such as "0 to 9", "10 to 19", etc.), and then plotted as bars. Each bar represents the number of samples for a range of data. Add the method below to your class from Lab 4:

```
std::vector<unsigned int> getHistogram(unsigned int aNumberOfBins) const;
```

(it needs to be decalred in `MyVector.h` and implemented in `MyVector.cpp`).

As we are using floating-point numbers to store values, we need to define how many bins we want to use. In the implementation of the method, you need:

- Create an empty `vector` of N elements (with $N =$ `aNumberOfBins`) and initialise every element to `0`. All can be done in a single call using the appropriate constructor of `vector`:

```
// Create an histogram with N bins initialised to 0
std::vector<unsigned int> p_histogram_data(N, 0);
```

- Specify the size of a bin. You need to consider the range of values in the array and the number of bins. Consider this example (Y_{noise} from last time): The min value is -0.83714 and the max value is 5.3959. So the range is $(5.3959 - -0.83714) = 6.23304$.
  - If there is 1 bin, its size is $6.23304/1 = 6.23304$. It starts at -0.83714 and finishes at $-0.83714 + 6.23304$.

  - If there are 2 bins, their size is $6.23304/2 = 3.11652$.
    * The first bin starts at -0.83714 and finishes at $-0.83714 + 3.11652 = 2.27938$.
    * The second bin starts at 2.27938 and finishes at $2.27938 + 3.11652 = 5.3959$.
  - If there are 3 bins, their size is $6.23304/3 = 2.07768$.
    * The first bin starts at -0.83714 and finishes at $-0.83714 + 2.07768 = 1.24054$.
    * The second bin starts at 1.24054 and finishes at $1.24054 + 2.07768 = 3.31822$.
    * The third bin starts at 3.31822 and finishes at $3.31822 + 2.07768 = 5.3959$.
  - If there are N bins, their size is $6.23304/N$.
    * The first bin starts at -0.83714 and finishes at $-0.83714 + 1 \times 6.23304/N$.
    * The second bin starts at $-0.83714 + 1 \times 6.23304/N$ and finishes at $-0.83714 + 2 \times 6.23304/N$.
    * The last bin starts at $-0.83714 + (N-1) \times 6.23304/N$ and finishes at $-0.83714 + N \times 6.23304/N$.
    * To generalise: The i-th bin starts at $-0.83714 + (i-1) \times 6.23304/N$ and finishes at $-0.83714 + i \times 6.23304/N$.

- Count how many values fall into each interval. To do so, for each element of the array, find the index of the bin corresponding to its value. Then increment the bin counter.

Task 2: Save the histogram

Add the following method to your class:

```
void writeHistogram(unsigned int aNumberOfBins, const std::string& aFileName)
    const;
```

Save the histogram as an ASCII file using an `ofstream`. You will need to include `<fstream>`. To declare a variable of this type and open an output file, just type:

```
std::ofstream output_stream(aFileName); // Open the file
```

Error checking is important, make sure the file is open before writing in the stream.

```
if (!output_stream.is_open()) // The file is not open
{
        std::string error_message = "Cannot␣open␣file␣(\"";
        error_message += aFileName;
        error_message += "\").";
        throw (error_message); // Throw an error
}
```

`ofstream` is extremely similar to `cout`. The only difference is that the data is stored in a file using `ofstream`.

```
output_stream << a1 << "␣" << b1 << endl;
output_stream << a2 << "␣" << b2 << endl;
output_stream.close(); // Close the file
```

In the case of the histogram, make sure the file contains an header. For this purpose, the first line will contain: `\"Min bin value\" \"Count\"` Then populate the file. Below are examples of output with different number of bins for the file `y_noise.mat` (file that we used last time).

1 bin

"Min bin value"	"Count"
-0.837137	401

8 bin

"Min bin value"	"Count"
-0.837137	109
-0.0580043	180
0.721128	31
1.50026	28
2.27939	12
3.05853	14
3.83766	14
4.61679	13

16 bin

"Min bin value"	"Count"
-0.837137	19
-0.447571	90
-0.0580043	124
0.331562	56
0.721128	21
1.11069	10
1.50026	12
1.88983	16
2.27939	9
2.66896	3
3.05853	6
3.44809	8
3.83766	5
4.22722	9
4.61679	7
5.00636	6

Task 3: Plot the histogram

The resulting file can be loaded in Matlab or Octave to produce a graph that can be included in reports.

Using Matlab/GNU Octave

Listing 1: Matlab script to plot the histogram.

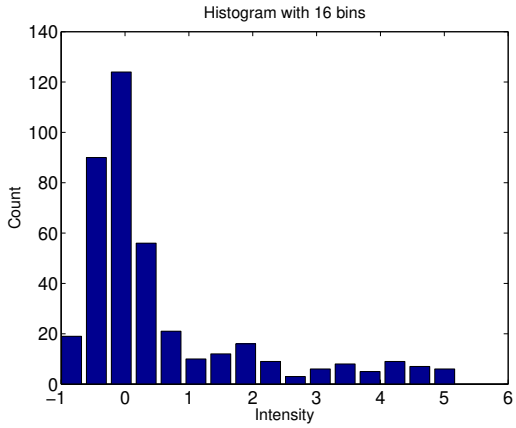
```
% Import the data from the file
histogram=importdata('histogram_y_noise_16bins.txt', '␣', 1);

% Plot the data using a bar chart
bar(histogram.data(:, 1), histogram.data(:, 2));

% Set the title of the graph
title('Histogram␣with␣16␣bins', 'fontsize', 16, 'Fontname', 'Helvetica')
% For GNU octave, use:
% title('Histogram with 16 bins')

% Set the axis labels
xlabel('Intensity', 'fontsize', 16, 'Fontname', 'Helvetica')
ylabel('Count', 'fontsize', 16, 'Fontname', 'Helvetica')
% For GNU octave, use:
% xlabel('Intensity')
% ylabel('Count')

% Store the graph in an EPS file (vector graphics)
print -depsc histogram_matlab.eps
% For GNU octave, use:
% print -deps -color -FHelvetica:16 "histogram_matlab.eps"

% Store the graph in a PNG file (bitmap)
print -dpng  histogram_matlab.png
% For GNU octave, use:
% print -dpng -color -FHelvetica:16 "histogram_matlab.png"
```

Listing 1 will use Matlab or GNU Octave [1] to load the file `histogram_y_noise_16bins.txt`, plot the histogram as a bar chart, and produce two files, an Encapsulated PostScript (EPS) file and a Portable Network Graphics image (PNG) file. Figure 1 shows the corresponding plots. You can find the script in `data/plotHistogram.m`.

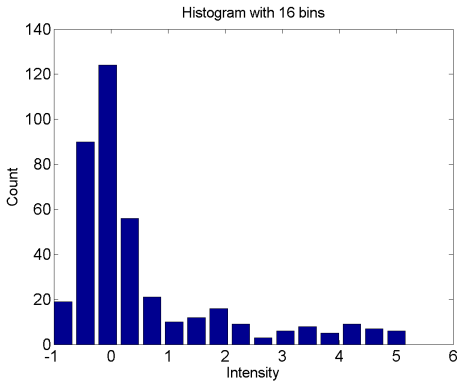
Using Gnuplot

If you do not have Matlab nor GNU Octave on your personal, then you can download Gnuplot. It is a lighter program dedicated to plot data. It can be found at `http://www.gnuplot.net/`.

[1] GNU Octave is an opensource project that aims at implementing a free version of Matlab.



(a) EPS file.



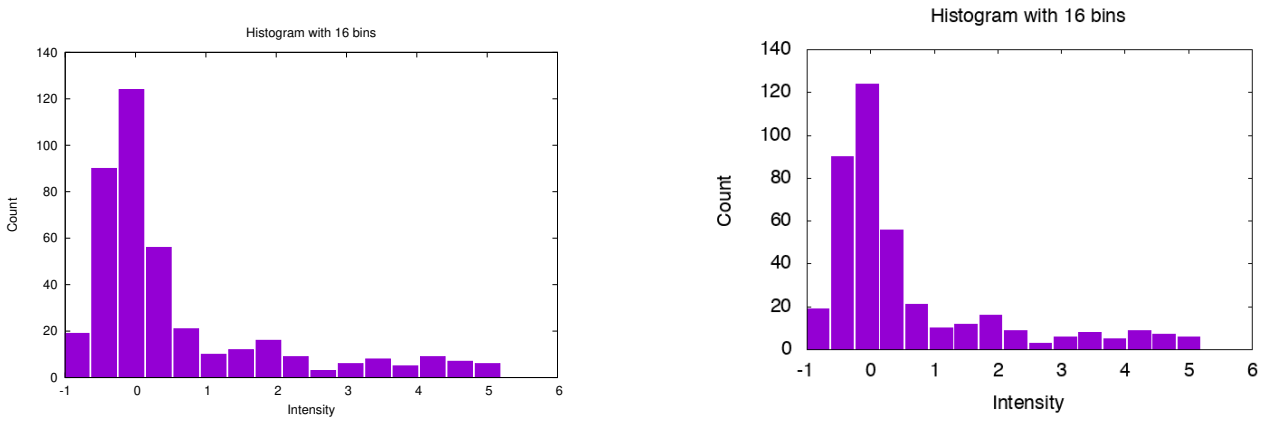
(b) PNG file.

Figure 1: Bar graphs generated using Matlab.

Listing 2: Gnuplot script to plot the histogram.

```
# Set the title of the graph
set title('Histogram with 16 bins')

# Set the axis labels
set xlabel('Intensity')
set ylabel('Count')

# Set the size of a bar in the bar chart
set boxwidth 0.35

# Set the style of a bar (use solid colour)
set style fill solid

# Make sure the output will be an EPS file
set term post enh color eps font 'Helvetica,16'

# Set the range of x values
# (note that it is optional)
set xrange[-1:6]

# Set the name of the EPS file
set output("histogram_gnuplot.eps")

# Plot the bar chart from the file
plot "histogram_y_noise_16bins.txt" using 1:2 with boxes notitle

# Make sure the output will be a PNG file
set term png font 'Helvetica,16'

# Set the name of the PNG file
set output("histogram_gnuplot.png")

# Replot the chart
replot
```

Listing 2 will be used with Gnuplot. It will load the file `histogram_y_noise_16bins.txt`, plot the histogram as a bar chart, and produce another two files (see Figure 2). You can find the script in `data/plotHistogram.plt`.



(a) EPS file.

(b) PNG file.

Figure 2: Bar graphs generated using Gnuplot.

Notes on the output files

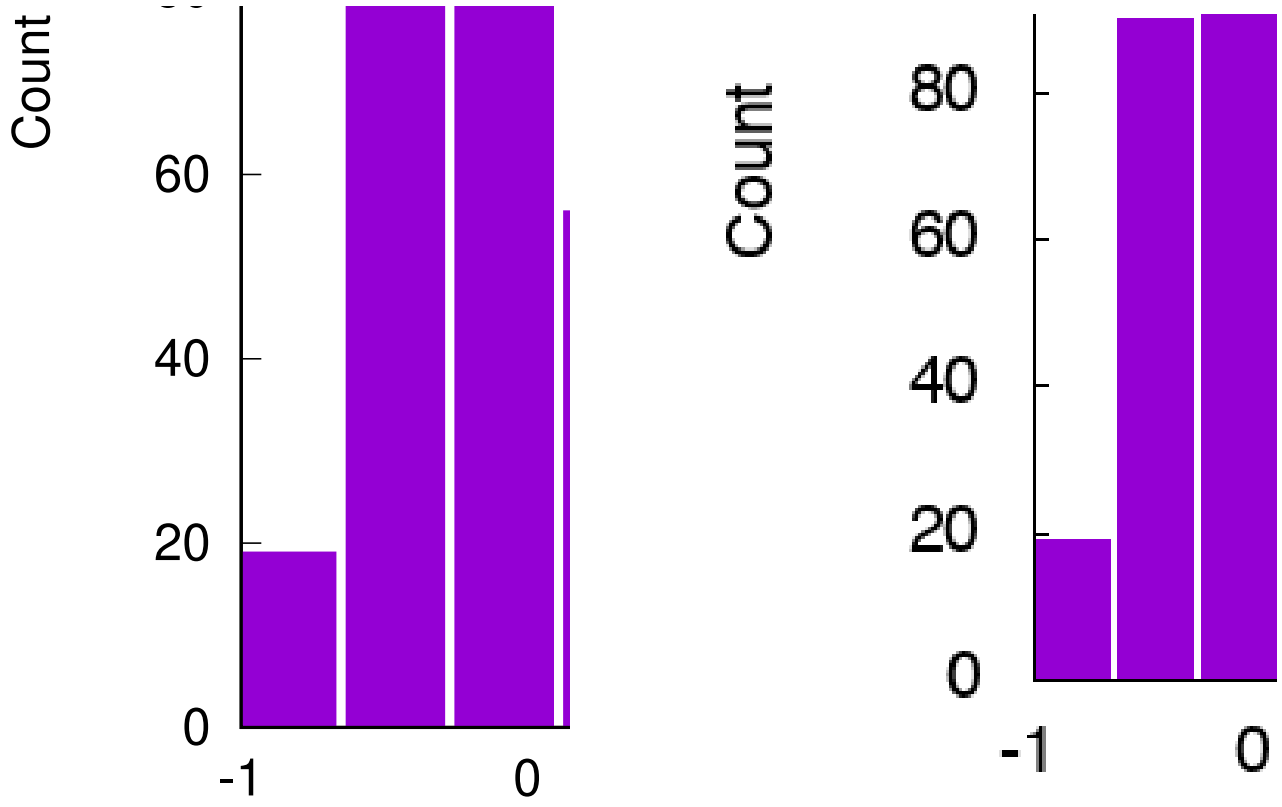
The EPS file contains vector graphics and can be zoomed in indefinitely (see Figure 3(a)). It can be converted as a PDF file (using `epstopdf` command from the console) and included in reports. The PNG file is a bitmap. If you zoom in, it will appear pixelised (see Figure 3(b)). It is suitable for web pages.

Summary

Today, you saw how to:

- Compute the histogram of a 1D Array. It is exactly the same for 2D arrays;
- Plot high-quality graphs.

You will re-use similar technique in your next assignment.



(a) EPS file.

(b) PNG file.

Figure 3: Close up of Figure 2.