Visualisation library

2. Description

Visualisation library allows users to recreate visual demonstration of numerical data by exploiting function of vslib. According to the level of difficulty, library consists of only functions that make simple line plots. Library also includes example application and the basement for other types of plotting, for example bar plot, as it was mentioned in the assignment. The level of difficulty is intermediate.

3. Instruction manuals

Download the project with all packages it includes. In order to utilise given functions, user has to upload vslib.plot and import Plot, Lineplot and Barplot. Once user has an access to them, she must to import csv-file on her directory. Only after completing these tasks user can proceed to visualisation. Major function has five inputs.

```
Lineplot('name', 'xlabel', 'ylabel', grid, ‘title')
```

Where **‘name’** is the name of input file, an absolute path of data, **‘xlabel’** is labelling for x-axis, **‘ylabel’** for y-axis, **grid** is a positive integer for adjusting plot’s grid and **‘title’** is simply the title of plotted window.

In the case if user has no desire to give additional options for her plot, she can mark None and 0 as these options return these features by default:

```
Lineplot('name', None, None, 0, None)
```

In addition, we are able to exploit example application. To do so, user needs to write down simple code:

```
Plot(‘example')
```

4. Libraries

Bunch of libraries are imported from PyQt5 as it plays major role in visualisation. The list of imported parts is showed below:

```
from PyQt5 import QtWidgets, QtGui, QtCore
from PyQt5.Qt import QColor, QGraphicsTextItem
from PyQt5.QtGui import QPainter, QPen
import sys
from PyQt5.QtWidgets import QApplication, QGraphicsItem,
QGraphicsScene, QGraphicsView, QGraphicsPolygonItem
from PyQt5.Qt import QRectF, QGraphicsLineItem, QPainter,
QGraphicsScene, Qt, QColor, QGraphicsPolygonItem
```

5. Structure of the program

- class **Reader**

Class Reader extracts information from numerical data, figures out wrong inputs and records new values. This class is extremely import for whole project as it becomes a helping hand through visualisation process in **GUI class**. Simultaneously it also gives us clear understanding where typo came from, i.e user can easily identify the problem during processing input data. Therefore the class is connected with **Error class**. Method def **read_csv_file**, major function of the class, returns six outcomes: minimum and maximum values of first and second parameter as well as differences between these values.

- class **Error**

Superficial class that raises an error if one has been caused. Error is accompanied by corresponding message.

- class **GUI**

This class is responsible for implementation of PyQt5 libraries. That is, it subordinates methods, such as: initialising window, title, labels and legend, setting grid and scales, drawing axises and lineplot. Finally, **GUI class** became a backbone for **Lineplot**, which makes further expansions (for example, development of **Barplot)** inconvenient. These inconveniences are described in chapter 10.

- class **Test**

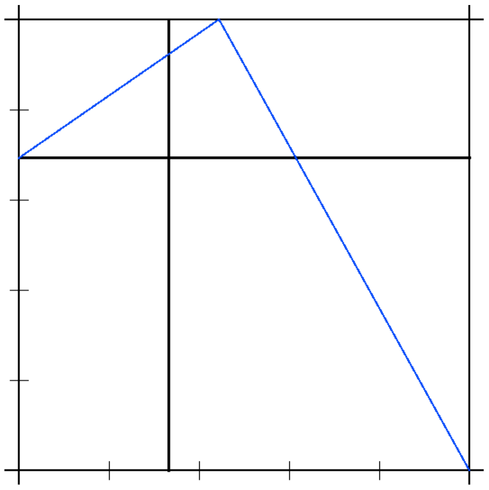
This ancillary class helps to find out dysfunctions, errors, wrong inputs or values.

- class **Plot**, class **Lineplot (Plot)**, class **Barplot (Plot)**

These particular classes allow us show up **GUI class** with its corresponding properties, i.e. plot the figure. It also provides extensibility for development of **Barplot**.

6. Algorithms

To proceed any further in visualisation, one must detect values using max() and min() functions in Reader class. For clarifying visual information, local maximum and minimum of the first parameter are located on top right and top left side of x-axis, respectively. Thus, in the window of size 500 x 500 these points take place at (0, y) and (500, y). For y-axis it works the other way as maximum locates on the top, (x, 0) and minimum at (x, 500).



In particular it means that whole data will be plotted in figure of size 500 x 500 pixels. Taking this circumstance into account we then can recognise 8 positions or modes, where location of axises varies. Bold lines are x and y axises:

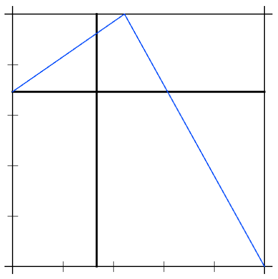


Fig.1 Mode (1, 1)

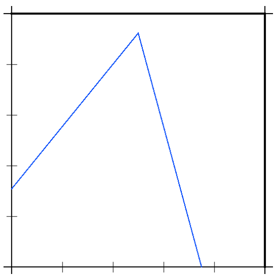


Fig.2 Mode (2, 2)

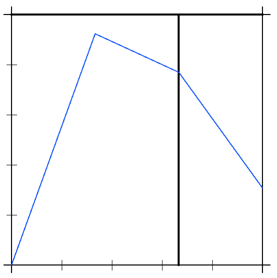


Fig.3 Mode (1, 2)

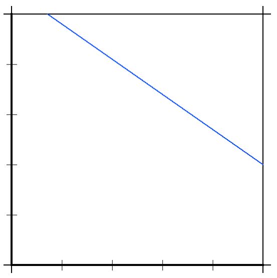


Fig.4 Mode (0, 0)

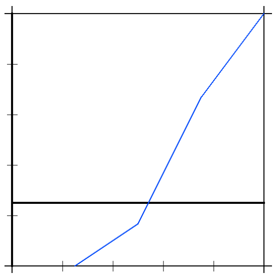


Fig.5 Mode (0, 1)

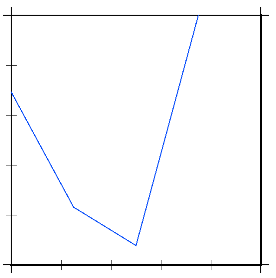


Fig.6 Mode (2, 0)

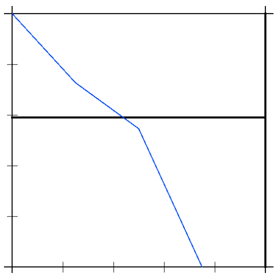


Fig.7 Mode (2, 1)

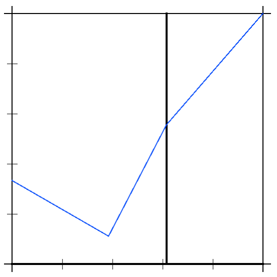


Fig.8 Mode (1, 0)

Here figures were identified by modes. First term of mode is related to x and second one to y, while **‘0’** means positive surface, **‘1’** is both negative and positive and **‘2’** is negative surface of coordinate system. Which mode is proper one can be easily computed by manipulation of points returned from **Reader class**. Terms of mode also correspond to location axises on plotted window. That is, knowing which mode corresponds to the data, one is able to 1) draw axises, 2) draw the line from numerical data and 3) write down proper segments. It is worth to mention that each mode has its own algorithm for line and scale calculus. For example, algorithms for line plotting with Mode (0, 0) that corresponds to Fig.4 is:

```
if MODE_X == 0 and MODE_Y == 0:
     line = QGraphicsLineItem(float(list[k-1][0])*500/x_max,(1-
float(list[k-1][1])/y_max)*500, float(list[k][0])*500/x_max,(1-
float(list[k][1])/y_max)*500)
```

Where list[k-1] is location of previous data point and list[k] is location of current one. For labelling segments of axis, following algorithm has been used:

```
k = 0
while k!= 6:
   if MODE_X == 0 and MODE_Y == 0:
       d1 = k * x_max/5
       text1 = QGraphicsTextItem(str(round(d1,2)))
       text1.setPos(d1/x_max*490,520)
```

Thus, all graphics are divided by 6 equivalent parts, where the first one has the smallest value (or in case of Mode (0, 0) it has 0) and the last one with the largest. Each algorithm is based on geometrical calculations.

7-8. Formats

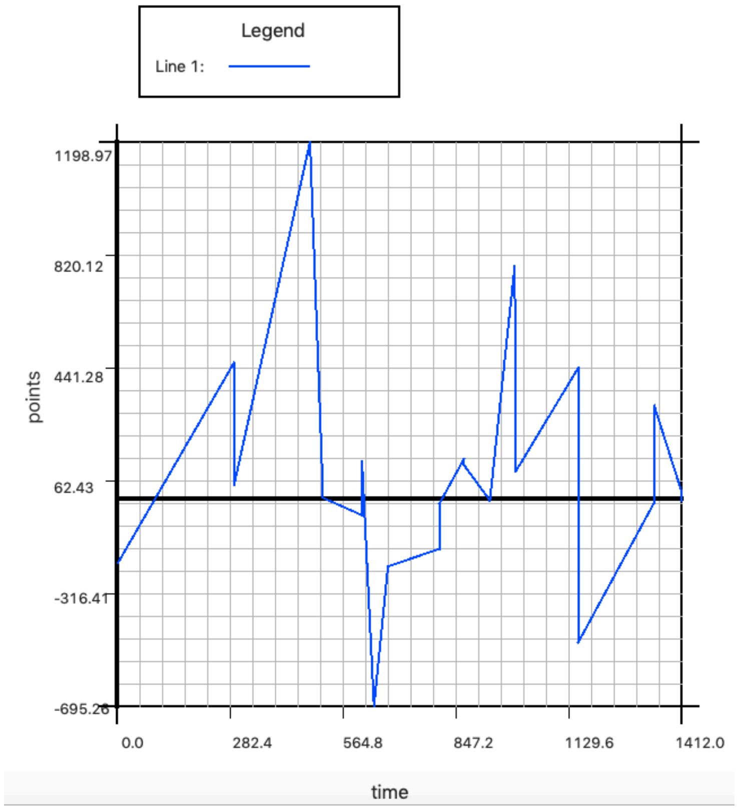
Program utilises comma separated values as an input file only. **Reader class** detects whether this data is correct in terms of structure. Unfortunately, at this stage **Reader class** has no launchpad for detecting other files but csv. As mentioned before, program expects R x 2 dimensional matrix, where R is number of rows, i.e. number of data points. The first column parameter is associated with X and the second parameter with Y, respectively. Once opened, data should be read and closed after exercising of all data points.

9. Testing

Testing had been provided by creating a bunch of csv files with correct and wrong inputs. The goal was to examine whether input data meets all requirements in forms of structure, correctness and comprehensiveness. All major problems had been tested. However, one still may expect failures due to specifics of csv files.

10. Disadvantages

The most sensitive disadvantage of the project is that sometimes numbers of the plot are very inconvenient to read. Indeed, going through methods in **GUI class**, we can confirm that parameter of each section is only the the difference between maximum and minimum value divided by number of segment. As we can see from the picture, both x and y axises are equally sliced by 6 segments. Moreover, grid as well as its mutability only depends on size of icon but not on graphic itself. As a conclusion, it is still complicated to make any exact outcomes from plots due to those disadvantages. In order to fix them, we substantially need to change calculation methods made in **GUI class** and add robust methods for number approximation. Algorithmically program would be more sophisticated than current version.



11. Top 3 of the program

+ Whole graphic can be seen no matter where it takes place in coordinate system, making the plot visually understandable.
+ Parameters of the plot can be easily adjusted on user's own.
+ Mode system. Only relevant parts are shown, no empty space.

- Numbers of coordinate systems are inconvenient to read.
- Grid doesn't work as wanted
- GUI class has too many methods. This fact makes extensibility more difficult.

12. Changes from original plan

More or less whole project went accordingly to original plan except the fact that now user has to sign all parameters into one line to perform a plot. Originally user could make changes on already plotted figure. In addition, instead of text files current version of program handles csv files.

13. Changes from original timetable

Project has not run on track of original timetable as too many delays have been occurred.

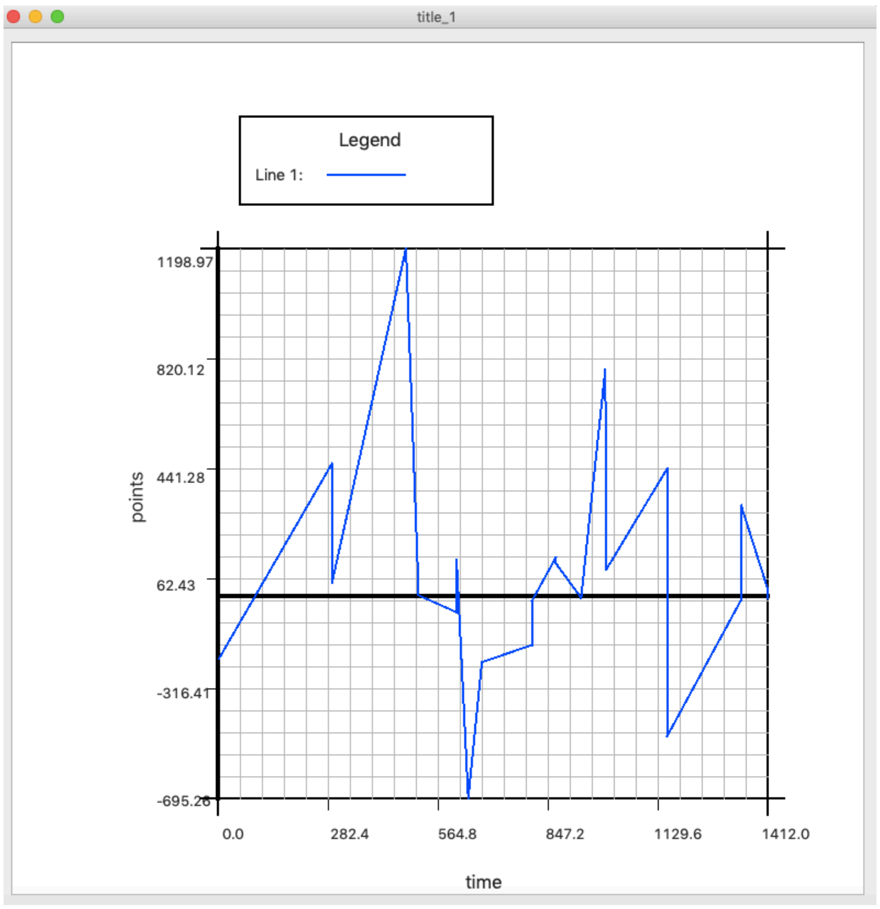
14. Conclusion

The goal of this project, recreate visualisation library, was achieved despite the number of disadvantages program has. Majority of tasks was also completed as user is able to configure labels, title and set a grid with a certain gap. Nevertheless, after several examinations it is clear that numerical part of plot is not optimised well: user can see the picture, but numerical interpretation leaves a lot to be desire. In this case grid plays only superficial role as it does not really assist to interpret a graphic. More sophisticated algorithms are required to fix these problems. In order to simplify the task of extensibility, skeleton of program could also be more well-structured.

15. References

https://doc.qt.io/qt-5/qtwidgets-index.html

16. Appendix


title_1
Legend
Line 1:
points
time