# Numerical data visualization library, project document

Jaakko Kilpi

903945

Tietotekniikka (Computer Science), 2nd year of studies

May 5th, 2022

## General description

The program that I have created is an application with a graphical user interface created using ScalaFX that allows the user to visualize a text file of a certain specific format as a line diagram of 1-n straight lines. Line diagrams are displayed on the top-right section of the window along with a grid. The other sections of the application window contain various buttons and textboxes with different functions that allow the user to control the application. The user can theoretically display infinite numbers of line diagrams simultaneously, but the program will eventually become slower and the section for individual diagram related buttons will become more cluttered as the amount of line diagrams currently displayed increases.

The project can be implemented in two different levels of difficulty, of which I initially chose the more difficult one. The more difficult project would have contained the capabilities to visualize histograms and pie diagrams on top of visualizing line diagrams. However, as I became sick near the deadline of the project, I was unable to complete the more difficult version of the project in time. Some aspects of this can still be found in the code, as various methods relating to histograms and pie diagrams are already implemented. Due to being sick, I ended up completing just the regular version of the project.

## User interface

The application is simple to open, as it merely requires that the user runs the GUIApp-object. This does require that the user uses sbt and IntelliJ, but as I have no intention of sharing this project with anyone aside from the course staff, converting the project to run from an executable file did not seem necessary. When opening the project, depending on the screen size of the user, the grid may not fit the window properly. The user can adjust the window to fit the grid properly, and this will not crash the program.

The user interface of the application is simple and easy-to-use. As described above, the top-right area of the user interface is dedicated for the diagrams and the grid. These can be controlled by buttons on the left side and under the diagram area. The buttons have labels on them, which give the user an idea of what each button does. By using these buttons, the user can toggle the grid, add a new line diagram, change the color or name of a line diagram, change the size of the squares in the grid, zoom in or out or reset the entire application to its default state. The user is also able to use the text boxes available to name the entire diagram or its axes and units of said axes. The user interface may look somewhat unrefined, but this allows the user interface to be simple to implement, and makes the program easier to run, even if the benefits are minimal.

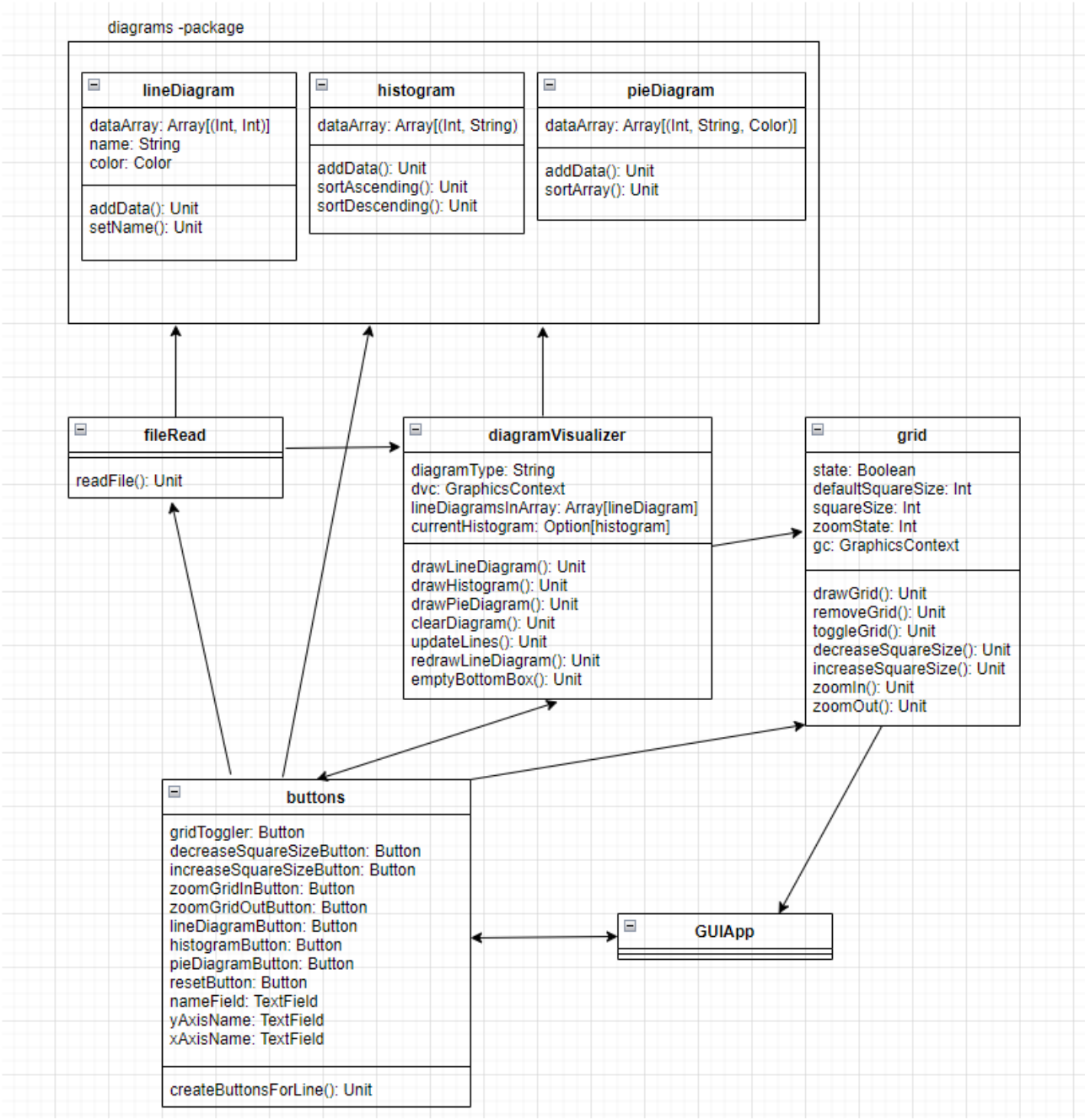
## Program Structure

The main program is split into three main packages, each of which contain different sorts of parts of the entire program. One package contains classes for the three diagram types, of which only the line diagram class ended up being used. The other diagram classes are still essentially fully implemented.

The “utilities”-package is another one of the three packages. This package contains the FileReader-object, which contains a method used for reading files which give input for the main program. There was no requirement to separate any of these objects and classes into different classes, but it makes the project look significantly clearer.

The final package, which is the “GUI”-package, contains most of the main functionalities of the program. These are separated into four different objects. Splitting the functionalities into four objects

may not have been necessary, but it did make the project much easier to work on. However, these objects are quite interconnected, and many of them interact with each other in ways that make mapping out their specific interactions rather difficult.




Here is a general UML-graph of the structure of the project. The GUIApp-object is left empty, as its various values and variables are only used because of ScalaFX and are irrelevant when looking at the project more generally.

## Algorithms

The algorithms used in the program are all simple and easy to understand. They are also accompanied by various comments describing their purpose. The most complicated algorithms used in the project are the sorting algorithms included in Scala, so I did not have to design them myself. I though Other more in-depth

functionalities of the program, such as adding new line diagrams, are handled using for- or while-loops and simple mathematical calculations, such as multiplications and divisions. In a few cases a match-case structure is used, but this could have been implemented in another way as well.

## Data Structures

The data structures used in the project are mostly simple, except for the diagram classes, of which only one ended up being used in the program. The class for a line diagram consists of an array of coordinates, two variables meant to store a string and a color, and two methods meant to either add new pairs of coordinates or change the name of the diagram. Other used data structures are very simple, and consist mostly of single strings, integers or arrays. The data structures of the program are not all mutable, but many of the important values, such as the “zoomState”, can be changed with buttons provided to the user in the user interface.

## Files and Internet access

The program uses simple text files which must follow a very specific format. In a text file, rows of text must only contain two integers separated by a comma, except for comment lines. Comments are declared by using a hash symbol, followed by whatever text on the row that the user wants to write. These rows are ignored by the file reading algorithm. The data can be in any order that the user wants, because the datapoints are sorted by the x-coordinate when adding them onto the line diagram class.

```
# Hash symbol is used for comment rows
# Data is separated by a comma (,)
# Data can be added in any order. Obviously, having datapoints with the same x-coordinates will not work.
3,6
5,4
6,7
100,400
```

The above picture is an example of a text file which is readable by the program, and it is included in the project. The project does not require internet access for anything and is fully functional when offline. The program can be used to read files, but it cannot create new files. As the coordinates of the line diagrams cannot be changed in the program, there is no reason to have the functionality to save any data in the form of creating new files.

## Testing

Testing the program was something I did very often. I did the testing mainly by running the program after any changes and seeing if it would work in the intended way. Because I have some previous experience with programming, testing was often unnecessary, but I do not regret testing a lot. Doing these sorts of tests is simply engrained into my mindset when I am programming.

The project also has some unit tests, included in the test-folder. These are simplistic tests for the general functionalities of the three diagram classes. Implementing some of these unit tests ended up being unnecessary, but I still included them in the project to show that the classes which ended up not being used still were implemented properly.

I did not have a specific test plan while developing the program, as I considered having one unnecessary. One could argue that my principle of heavy testing is a sort-of test plan in and of itself, but I did not think of it as such.

## Known bugs and missing features

The program does not have many bugs that I am aware of. One important bug, which may come to mind when testing the program is to read a file that is not of the correct format. When doing this, the program throws some errors, but this does not crash the program and allows the user to continue uninterrupted. Other bugs and errors were something I considered when developing the program and ended up not being there in the final version of the program. An example of a bug that I found out when developing the program is that when the user would not choose a file when creating a new diagram and would instead close the file choosing window, the program would stop working.

The main missing features of the project are the two other diagram types: histogram and pie diagram. These are partially implemented, but due to being sick I was unable to finish the program with these included. Another missing feature that I originally thought of implementing was the ability for the user to do more in-depth edits to the diagrams using the program. This turned out to be unrealistic to implement, and it was not required in the guidelines.

## 3 best sides

1. The program is easy-to-use. The functionalities of the various buttons are very apparent, and the user should not have any trouble using the program, if they have understood the also simple file format.
2. The program has very few bugs. I have not yet found many bugs when testing the program. All the buttons and functionalities associated with the buttons are programmed in a way that makes them not buggy. The simplistic nature of the various algorithms helped with this.
3. The user can have multiple line diagrams displayed at the same time. I initially misread the project guidelines and thought that this was a requirement for the project. It was only after implementing this that I realized that it was not. However, this is something that I think I would have implemented regardless, as line diagrams often have multiple different lines describing different things.

## 3 weaknesses

1. The visual style of the program could use some work, as it is not exactly pleasant to look at. This was not a priority for me. Fixing this could be done in various ways, but it would require some work.
2. The missing diagram types. If I had some extra time or if I was not sick, I would have implemented these as well.
3. The bug that happens when the program reads a file that is not of the correct type. While not particularly important in terms of the program working, this is still something that I would have fixed, if I had additional time.

## Deviations from the plan, realized process and schedule

I did not stick to my original plan of working for one hour each day at all. Perhaps this plan I set for myself was unrealistic, considering I tend to work on a single project or exercise in much longer bursts, which often consist of my entire day. This allows me to become much more acquainted with a project while I am working on it. The time estimates I had initially set turned out to also be unrealistic, as I required two additional weeks of time. One week was because of other university projects taking up my time, and the other one was because I was sick. If I had worked on the project much earlier, I could have avoided these additional weeks.

The actual plan for the project was something that I discarded rather quickly as I became more accustomed to ScalaFX. I instead just began to work on a simple example found on the "Additional resources" section of the A+ page of the course. Due to my previous programming experience, I did not encounter many issues when developing the program, and I most likely could have finished the project within the first week if I was willing. However, I did not initially understand this, and have only come to this realization after I finished the project. Becoming sick was something that I did not initially anticipate, and it affected the development process. The one-week additional expansion was not exactly enough to cover the entire time of being sick, as I am writing this while still being ill, but I figured I would have to finish the project eventually, so I began working on it again regardless. Overall, I can say that I have learned a lot from my personal capabilities as well as developing projects in general while developing this program.

## Final reflections on the project

Personally, I felt somewhat disappointed with the final state of my project. I really hoped that I would have been able to finish the more difficult version of the project, but alas, my time working on the project became much more difficult when I became sick. However, I do think the program does what it is supposed to do very well, and I am proud of what I did accomplish.

As for the program itself, I think that many of the data structures and algorithms implemented are logical and well-functioning. I could have optimized them in some ways, but this just did not seem necessary to me, as students of the course are not expected to have taken the Data Structures and Algorithms course. I have taken this course, but I was disappointed that I was not able to use the things I learned during the course in this project. Perhaps some of the other projects could have been more suitable for this purpose.

The main thing I would do differently if I were to do a similar project in the future is that I would start working intensively on the project right from the start. This would have allowed for more leniency in terms of changes to my schedule that I have not been in control of, such as becoming sick.

References, which proved to be useful:

The A+ page for the course, https://plus.cs.aalto.fi/studio_2/k2022/

ScalaFX tutorials uploaded to YouTube by Mark Lewis, https://www.youtube.com/c/MarkLewis

ScalaFX documentation, https://appdoc.app/artifact/org.scalafx/scalafx_2.9.2/1.0.0-M2/scalafx/package.html