Project Document

1. **Personal information**

Title: Fitting and Visualization of a Regression Model

student number: 715683

degree program: Energy and environmental engineering

year of studies: 1st year

date: 20.4.2020

2. **General description**

A general description of what has been created. This section can in most cases be copied from the project plan. If there have been substantial changes compared with the plan, these should be mentioned here. If it is possible to implement the topic in different levels of difficulty, state also on what level you think the work was finally completed.

Fitting and Visualization of a Regression Model

Theme: statistics

Build a program which can be used to fit a regression model to a dataset and visualize the results.

Regression means estimating the relationships between variables. Perhaps the simplest, yet very useful method for doing this is simple linear regression. It can be used when the data has two variables (x, y) and we can reasonably claim that they are proportional to each other, i.e.

y = a + b * x

for some a and b – finding appropriate values for them is the “fitting” part of this project.

The formula can be useful even if the variables are directly proportional, because it is often possible to find a function which transforms the x variables so that they are. After doing this, the formula becomes

y = a + b * f(x)

For example, if the true shape of the data is a parabola instead of a straight line, we could take the square root of x before fitting the model.

Requirements

Intermediate

- The user of the program should be able to load the dataset from a file of their choosing. The file may contain even a large amount of (x, y) coordinate pairs.
- Implement simple linear regression. (Only the first equation. Since this is a programming course, not a math course, it is not necessary to show why your method produces a good fit – if you have not taken MS-A050X First course in probability and statistics or have equivalent knowledge, you may study the subject online or ask an assistant).
- The program should produce a plot of the original datapoints and the fitted model. The appearance of the graph, such as the endpoints of the axes, can be configured by the user.
- The implementation should be done with extendability in mind; when designing your program, you should consider how you could make it as easy as possible if you were to implement the advanced level requirements in the future.

Advanced

- All the requirements of the intermediate level.
- Support for loading at least two different file formats.
- Implement at least one regression method in addition to simple linear regression. (Either both the first and second equations, or the first equation and a method of your choice).

Important: My project does <u>not</u> work with the Java versions from AdoptOpenJDK.

It <u>does</u> work with the java.com versions.

For Windows: Version 8 Update 251

For Linux: Linux x64 Version 8 Update 251

https://www.java.com/en/download/

The link should direct you to the previously mentioned java. If the link does not work or the Java versions have changed, you can probably just google "download java" and choose the first result which is java.com. It should work on java.com versions. On my home computer, it also worked with the Oracle version. I did not test it on Linux though.

My project is implemented according to the advanced level. My project can handle csv, tsv and txt files. You can even choose the delimiter yourself. As a regression analysis only handles numerical data, those file formats seemed to be the most natural choices.

I implemented segmented linear regression as the regression method of my choice in addition to simple linear regression.

3. User interface

Introduction to how to use the program: How is it started? What can you do with it? What commands the user can give and how, etc.

The program can be started through Scala, by running the "Plotting" file (inside the regression folder) as a Scala application.

You can visualize paired data (x, y) by plotting it on three different plots. When the program is started, some example data is visualized. In addition to that, you can open your data files and visualize them.

One plot includes the separate data points as a scatter plot. Another plot has a simple regression line that best fits the given data. The third has a segmented regression plot where there are many regression lines successively and it visualizes local trends in the data. All three plots are visualized at the same time.

In addition to opening new files for plotting, you can change the delimiter according to the data you are using.

You can also modify the view by changing the labels of the axes, the size of the plots, axis ranges, tick size and moving the plots around. For the segmented regression, you can choose how many datapoints you want per segment, essentially choosing how many segments you want.

For changing the value of the delimiter, segment sizes, labels, ranges, tick size, sizes, or moving the plots around, you type the value into the textbox that appears. For the axis range and tick size, you type in three values separated by a comma and for moving the plots, you type in two coordinates. For the others, you type in just one value. The values must be numbers, for all except labels and delimiters.

There are some restrictions on what values can be used, to not burden the program too much. The program can become slow because of massive graphics. Also, if there are more than 1000 data points, the program only plots 1000 of them with regular intervals.

You can also close the program through the exit button.

I used ScalaFX for my graphical user interface. There is no textual user interface, only a GUI.

4. Program structure

How the program is split into main sub-parts? What was the final realized class structure? What kind classes do you use to describe the problem domain of the program? What part of the problem each class is modeling? What are the relationships between classes? What classes are used to describe the user interface?

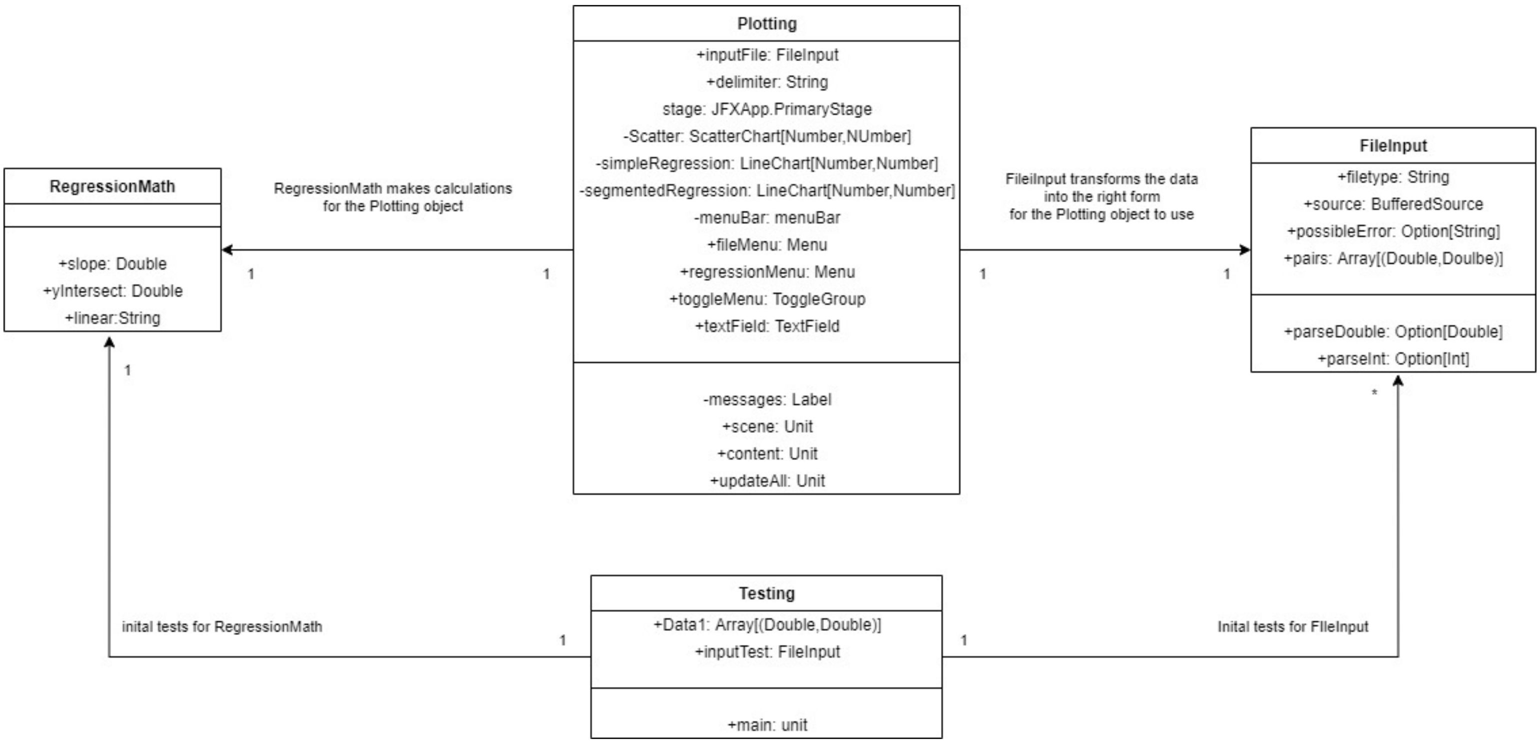
Please, include some kind of graphic class diagram, for example, UML class diagram, but other methods can be used, too. Introduce the key methods in classes. Note. The essential thing here is what the methods carry out, not how they work internally.

Here it is good to consider also possible other solutions and justify the decided structural solution.

My project is split into three files, plus the testing files. The RegressionMath file contains the “RegressionMath” object which contains all the mathematical formulas. There are no instance variables in that object, but there are methods that can be used by other objects. The key methods are “slope” and “yIntersect”. Those contain the simple linear regression formulas and they are used to create regression lines. The other methods are helper methods used for regression.

The Fileinput file contains class “FileInput”. You can create instances of that class each time you want to read a file. It reads the data and outputs it into an array.

The Plotting file which contains the object “Plotting”, handles all the heavy work. It functions as the graphical user interface as well as the application that can be run.


Plotting
+inputFile: FileInput
+delimiter: String
stage: JFXApp.PrimaryStage
-Scatter: ScatterChart[Number,NUmber]
-simpleRegression: LineChart[Number,Number]
-segmentedRegression: LineChart[Number,Number]
-menuBar: menuBar
+fileMenu: Menu
+regressionMenu: Menu
+toggleMenu: ToggleGroup
+textField: TextField
-messages: Label
+scene: Unit
+content: Unit
+updateAll: Unit
RegressionMath
+slope: Double
+yIntersect: Double
+linear:String
RegressionMath makes calculations for the Plotting object
FileiInput transforms the data into the right form for the Plotting object to use
FileInput
+filetype: String
+source: BufferedSource
+possibleError: Option[String]
+pairs: Array[(Double,Doulbe)]
+parseDouble: Option[Double]
+parseInt: Option[Int]
Testing
+Data1: Array[(Double,Double)]
+inputTest: FileInput
+main: unit
inital tests for RegressionMath
Inital tests for FIleInput
1
1
1
1
1
1
1
*

In my initial plan, I had a lot more classes, but now afterward it seemed better to have less. Still, I thought about splitting the Plotting file into two or three parts, but I figured there would be a lot of technical difficulties and work. Also, it could have made some parts less clear.

As mentioned, my final class structure contains fewer classes than my initial plan contained. Let's compare the planned structure and the final structure:

In the initial plan I had 9 classes: File handling, data handling, regression, plotting, commands, console output, exceptions, run and testing.

I merged the file handling and data handling classes into one "FileInput".

The Regression is "RegressionMath".

Plotting is as "Plotting"

The commands class is not the same as initially imagined. I was going to create a textual interface, but now when it's graphical, the commands are integrated inside the GUI using onAction methods. I believe I could move the onAction commands in the plotting class into its own class, but that could create problems of its own.

The console outputs and exceptions are generally included in the onAction methods. They could possibly be moved into their own class together with the onAction methods. I have no "throw" exception statements in my code. I'm handling different exception scenarios with if-else structures and reporting to the user accordingly, using the GUI.

Testing is "Testing".

5. Algorithms

A verbal description of the algorithms you use, i.e. how the program performs the necessary tasks. For example, how the necessary mathematical calculation are carried out (include formulas)? How does an algorithm to find the shortest route between two cities work? How does the artificial intelligence you designed for the game work? You can use diagrams as an aid according to your need. Where appropriate, present what other solutions would have been possible and justify your choices.

The purpose of this section is to explain the principles how the problem(s) at hand are solved, not how the algorithms were actually coded. Therefore, no descriptions of classes or methods or any other Scala or program code issues should be included here. However, a pseudo-code presentation of the most important non-wellknown algorithms would support the verbal description.

NOTE! Every program includes algorithms; others might be simpler than others, and many of them were invented from the very beginning. Here it is good to describe some of the most important ones.

For plotting the regression lines, I have an algorithm that uses mathematical formulas. That algorithm decides the slope and y-intersect of the lines and the points where the line starts and ends. Mathematically the lines would be infinitely long, but my algorithm chooses the start and end points. As a start point it chooses the x-coordinate of the first data point of the group and as the end point, the x-coordinate of the last data point of the group.

The mathematical formulas used:

slope = (mean(x)*mean(y)-mean(x*y))/(mean(x)^2-mean(x^2))

y-intersect = mean(y)-slope*mean(x)

For the segmented regression plots, I have an algorithm for splitting the data into groups. Each segment contains the specified amount of points, except for the last one. The last one always contains the remainder. If the groups are not evenly split, the last group will be smaller.

Then there is the delimiter-choosing-algorithm. If it notices that your data doesn't contain the right delimiter, it checks if there is another delimiter that could work instead.

There is also on algorithm for choosing the most valuable data. If the data is larger than 1000, it chooses the best ones. It groups the arrays using an iterator and then taking values from regular intervals. This can be done, because the program sorts the data according to the x- value before plotting it. But, you must keep in mind that the algorithm is essentially an filter. It filters away the data that is sees as least useful. It means that all data is not plotted on the scatter plot and the segmented plots can greatly differ from plots that would include all data. The algorithm is made to restrict the graphically intensive features. The simple linear regression still uses all data points and is not affected by the algorithm.

I also have if-else statements which work as algorithms. They either affect the program in a meaningful way or report an exception message to the user.

Also, for removing bad data (non-numerical data etc.) there is an algorithm. I uses different filters and regular expressions

6. **Data structures**

What types of collections / data structures are used to store and process the data you need in the program? Why these structures? What other options would there have been? Are the structures mutable or immutable? If existing Scala collections are used, their exact definition needs not be presented. If some data structure is programmed by self, its working must be explained.

I used arrays for my project, because applying and updating values is performed in constant time. Also, arrays are the data structures I'm most familiar with. At one point I used lists as well, but I replaced them later. I did that to only stay with one type of data structure, to keep the program structure more simple. Arrays are used in their immutable form: I'm only using mutuable object in the form of variables. I can update the arrays by updating the mutable variables associated to them.

7. Files and Internet access

This section describes what kind of files the program deals with, if any. For example, are they text files or binary files, and how is the data in them presented? Present the final file format at the level that the assistant can, if desired, easily create test data for the program (an example of file content could be a good idea). If the program needs to create user's own configuration files etc., include them as an attachment with the source code. The purpose is to submit the code in a form where the assistant can easily try it out without using much time for tuning up the program environment.

If the program accesses Internet to read/write data, e.g. web pages, explain the relevant things here as appropriate.

The files used for the program are textual files

The data used for the program must be in a specific format. The first row of the data file has the labels for the x and y axes. Every row after that has the data points. The labels and the data points are each separated by a delimiter, e.g. a comma.

Here is an example file, with a comma as the delimiter:

```
Dogs,Cats
1,2
3,4
5,6
7,8
9,0
```

The example file could represent the relationship between owned cats and dogs in the households of Finland. Each row represents one household and contains how many dogs and cats that household owns.

The filetype must be one of the supported types. The extensions are important, unsupported extensions won't work. The file has to be named with the right extension. E.g. data.csv. By extension, I mean that the file name ends in: ".csv". There are test files in the project you can take a look at.

The project does not depend on an internet connection.

8. Testing

A description of how the program was tested. Was the testing process different compared with the planned testing process.

Does the program pass all the tests? How was the program tested during the implementation process? Were there something essential which was missing from the test plan? If unit tests were used for some part of the code, please, report this here.

Even though I have a runnable "Testing" object, the majority of all testing was executed in other ways.

First, I tried to get the unit test library to work, but I failed. Then, I created the object "Testing" where I could test things and I put all my tests inside the object. I did that using a "main" method, which I used in the first weeks of my project.

After I managed to get the GUI working, I started testing things directly by launching my app and looking at the results and the error messages. I ran it probably one thousand times. After each time I tweaked or fixed something.

Sometimes I also went with the tried and tested "println" statements. If something didn't work, I positioned "println" statements in strategic places of the code and tried to find the problem spot. That worked very well, and I usually found the problem.

By showing the program to other people, I found things to improve and bugs which I wouldn't have found alone.

I also tested the file with data from Tilastokeskus and the data can be found in the project as "Statfin.csv".

My program passed the majority of tests, but there are some bugs that exist, and I will discuss them next.

9. Known bugs and missing features

In this section, describe all the bugs and defects you know in your program. Tell how these problems could be fixed if you would continue with the project. *The less assistant finds problems/bugs in the code which you claim to work the better*. Thus be honest.

In the following sections, the project final outcome and the work process will be evaluated.

The program does not work with all Java versions. I tested the program on an Aalto Linux computer and it worked only if I downloaded another java 8 version. You have to download a proper version of Java on your computer, otherwise my program won't work. AdoptOpenJDK does not work. The oracle and java versions should work according to my tests. For the Aalto Ubuntu computer I downloaded Linux x64 Version 8 Update 251 from java.com. The oracle version also worked on my home computer, but I didn't test it on Linux.

If the data inside your file contains non-numbers, the delimiter is wrong or the data is weird is some other way, the plots can disappear. The slope of the regression lines can then result to NaN or infinite. Then the auto sizer can't find a good range for the axis, so it makes the range zero. You can bring the plot back to normal by adjusting the size of the axis/axes. I feel like I fixed it, but I'm not 100%. I'm reporting it as a bug, because I had the problem for so long and if it still persists, it is very disturbing.

The program can also throw error messages for some weird delimiters or data, but nothing else happens. By "weird", I mean some combination of non-numbers in the data file or a delimiter which doesn't act as a delimiter in the data. I have a filter that removes some problems, but I don't think it's 100% reliable.

The program could be slow when handling too much graphics, but I tried to prevent it, using restrictions.

On normal usage, my program should behave normally, but maybe my exception handling could have been better.

10. 3 best sides and 3 weaknesses

> The assistant will use plenty of time to familiarize with your program, but s/he does not necessarily see and know your implementation in the same way as you do. If there are any points in the program that you consider particularly good, please, mention here one to three of them with a short justification. If there are points in the program that you know are weak, you can also mention these. In this case, the possibility that these weak points dominate the evaluation, is greatly reduced. Here you can also present verbally, how these weaknesses could be corrected, if desired.

Best:

The best side is that the program looks nice with the colors and the graphs. I especially like the segmented regression with many colors and that there are many possibilities to modify the view.

I implemented some fool proofing using filters and regular expressions (regex). There can be some mistakes in the input, and it will still work. It removes non-numbers and chooses the right delimiter.

Although it doesn't show. I managed to remove a lot of excess code by combining things. For example, the radio buttons and the text field can be used for many purposes. In the code, they are only created once, but they are used later for different purposes. I did a lot of chopping and changing to remove repeating code.

Weaknesses:

The bug where the program can't decide a good range and the plots disappear is probably the most disturbing one. But, let us assume that I managed to fix it.

The user experience is not the best. I later realized how much better some things could have been done so that the user can more easily use the program, but I did not have time to complete everything. E.g. directly clicking the labels to change their names and dragging plots to move them.

The fact that I couldn't find internet data that worked well with my program is a bad side. I always had to clean the data beforehand. It seemed like they were all structured a little different, with small nuances conferring with my implementation.

Also, the Plotting file could be split into more parts. Perhaps then it would be easier for the reader to read, because there is a lot of code in that object.

11. Deviations from the plan, realized process and schedule

> Tell here in general terms about the order in which the project was finally implemented, for example, in two week granularity. Where did the process deviate from the initial plan and why? How well did the time estimate in your plan match with the reality? Where were the biggest differences? What about the order of progress? Did you possibly learn something worthwhile during the process?

My plan deviated quite much from the original plan. But because I had never done anything similar, the plan was very rough. Also, I did not know how the workload would be distributed in my other courses. Turned out I did not have much time in period 4 to work on the project. Instead, the whole 2 weeks following period 4 was used to work on my project. So, in the end, my prognosis of 160 hours of work was quite right. Maybe I used even more time.

I implemented all the mathematics of the project in the first weeks. Then I started to work on file reading/data handling. I got stuck very often with the first steps of the project. I was a bit disappointed that we had not gone through a lot of the basics of creating a project. First I got stuck because the program didn't treat my program as a Scala project and It didn't check the code. Turned out I needed some libraries for it to work.

Also, when trying to use libraries from the internet, I was totally lost. There was a lot struggle with that. Even now, I don't have a foolproof way to do it. Everybody on the internet assumed you had heard about SBT and Maven which I never had. This course should definitely go through how to use libraries. There was nothing about that. Thankfully, I received help from friends and course assistants.

I got very little done in the following weeks because of other courses. But I knew I would have time to work on the project on the exam week and (unfortunately) on the Easter holidays.

During that period, I implemented the Graphical User Interface from start to finish. That part I enjoyed because I found some good resources for how to use ScalaFX. It was also nice, because with very little code, you got a lot done.

Next, I started cleaning the code and added some more features. Earlier I had implemented support for csv files and now I implemented support for tsv and txt files as well as support for any delimiter the user wanted to use. I also tried to import some JSON library to handle JSON data, but I had problems again with importing libraries and I decided that there was more important stuff to fix.

I fixed some bugs and also had some customers (dad and brother) trying out my program and giving feedback on where I could improve. Based on that, I improved the feedback my program gives to the user.

Through all of this, I constantly tested my functions and my program. I didn't want a situation where I would have to delete everything because of some problem in the foundations. I found my errors more easily, the less changes I had made before testing.

As an amateur, I decided that I shouldn't plan too much and just start coding. I hade never done anything similar, so my planning wasn't really based on any experiences. My planning was only theoretical, because I didn't know what to expect.

During this project, I learned a whole bunch about programming in general and some specifics regarding my project. I learned some of the struggles/threats about programming as well as the opportunities.

12. Final evaluation

> ”Summary” and self-assessment, where you can repeat some of the above things.
>
> Evaluate the quality of the final program, tell about its good and bad aspects. Are there significant shortcomings and where do they emerge (a good explanation in the document can replace the small shortcomings)? How could the program be improved in the future? Could the choice of solution methods, data structures, or class structure be better? Consider how the structure of the program is suitable for making changes or extensions?
>
> Finally, evaluate what you would do differently if you started the project again from the beginning.

I think it is a good program if the data is in the right form. It does some of the basic things I would do myself if I would want to visualize and make a regression. It could be a bit easier to use.

The structure of the program is suitable for making more changes and I had many ideas, but with limited time I could not implement everything.

I tried to find a relatively easy plotting library, where I still would have to do things myself. The Swing library didn't seem optimal for plotting, so I chose the ScalaFX library. It worked very well for creating a GUI and plotting straight lines. But when it came time to do the polynomial regression, I couldn't find a way to do it in ScalaFX. That is why I chose to use a Segmented Regression instead. In the end it looked really nice. Also, it shows the same trends as a polynomial graph would. My implementation can show up to any order, just by increasing the number of lines. The segmented regression is constructed by grouping the data into suitable groups, and for each group plotting a separate regression line. The segmented regression plot is great for visualizing trends in specific portions of data, which aren't global trends for all the data.

In my user interface I could have improved the user experience, but the work could have been too demanding. I thought about implementing dragging abilities to change the order and size of plots. Also, I thought about making the plots clickable, so you directly could change the labels and data ranges by clicking them. I decided to focus on other, more urgent things. Later, I could make changes to make the project more user friendly.

On the other hand, if the ScalaFX library doesn't suffice, there could be some problems. My math and input files would still work for another plotting library, but the Plotting class would have to be redesigned. If I would transfer the user interface to another library, the whole user interface would have to be changed. But if only some parts would be implemented in the other library, it could be doable in reasonably less time. E.g. I would imagine implementing the Polynomial regression plot and opening it in a second window could be rather straightforward. I didn't want to do that though as it could have turned out as rather clumsy for the user. This would have to be investigated further for further development.

I would have liked to plot the data points (scatter) and the simple regression line in the same plots, on the same x,y-coordinate system, but I couldn't figure out how to do that. So, I had to work with what I got.

The Plotting file could possibly be split into more parts to make the class structure clearer.

If I would start the project from the beginning, I would try to make the program even easier to use and try to get a library to work for reading data files so I wouldn't have to read them myself. I would try to search for more data beforehand and see how it looks. Also, there seems to be a lack of good graphing libraries in Scala so I would maybe use another programming language to get access to other libraries. Importing libraries was a pain in the ass for me. Hopefully, I will get some sense of that later.

All in all, I'm quite satisfied with my program. There is always something to improve. I didn't expect any miracles for my first program.

13. References

What books, websites or other material you have been, excluding course online textbook?

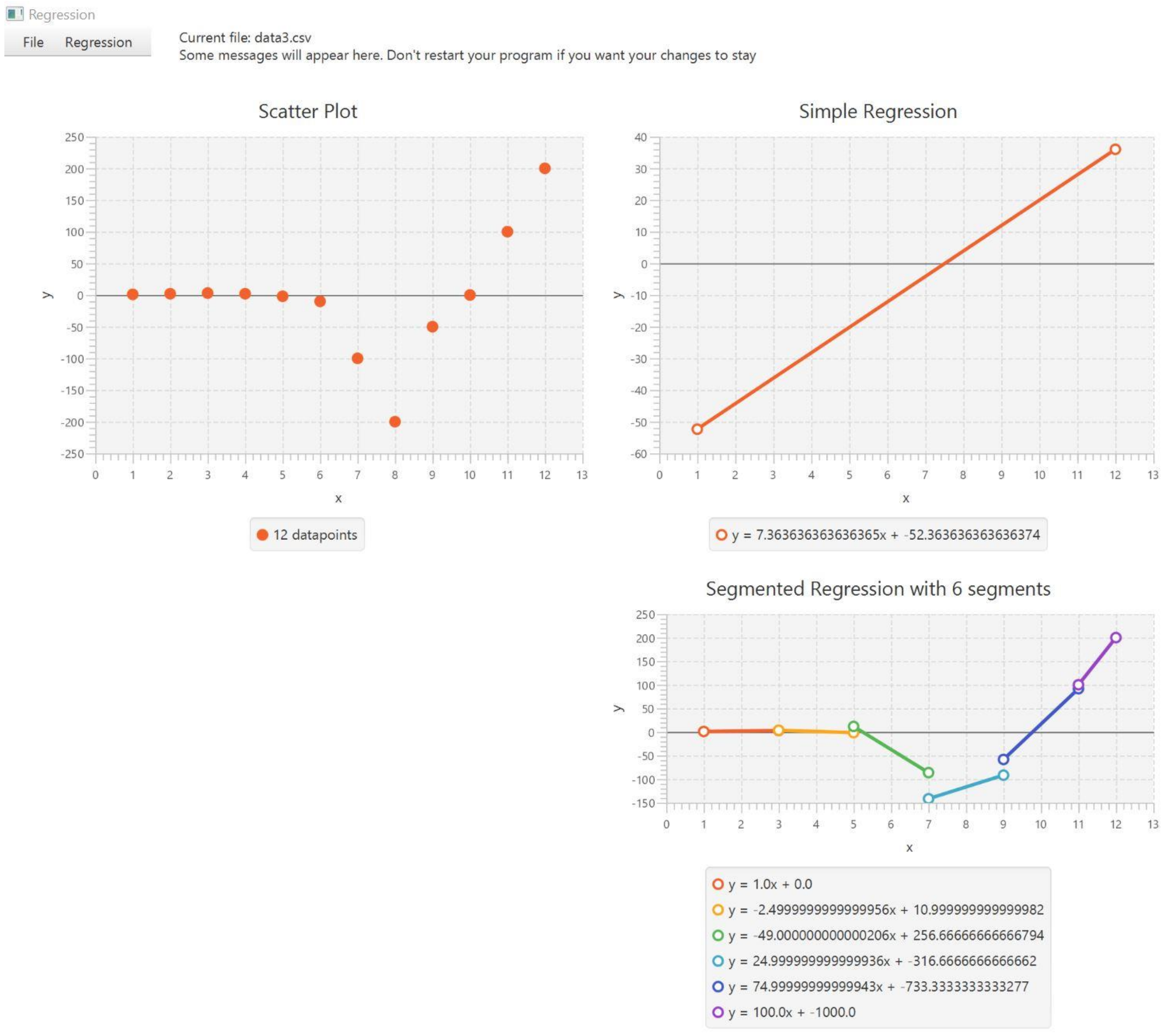
The Youtube videos by Mark Lewis on using Scala and ScalaFX were amazing. In addition to those, I used other youtube videos and stack overflow for specific errors or problems. For creating the GUI, I used the ScalaFX library API.

14. **Appendixes**

The most important project appendix is **the full source code of the project**.

Additionally, at least for text-based programs, you should add **A few illustrative execution examples.** With graphic programs, you can include screenshots of the different execution situations of the program (find out yourself how to do it on your own computer).

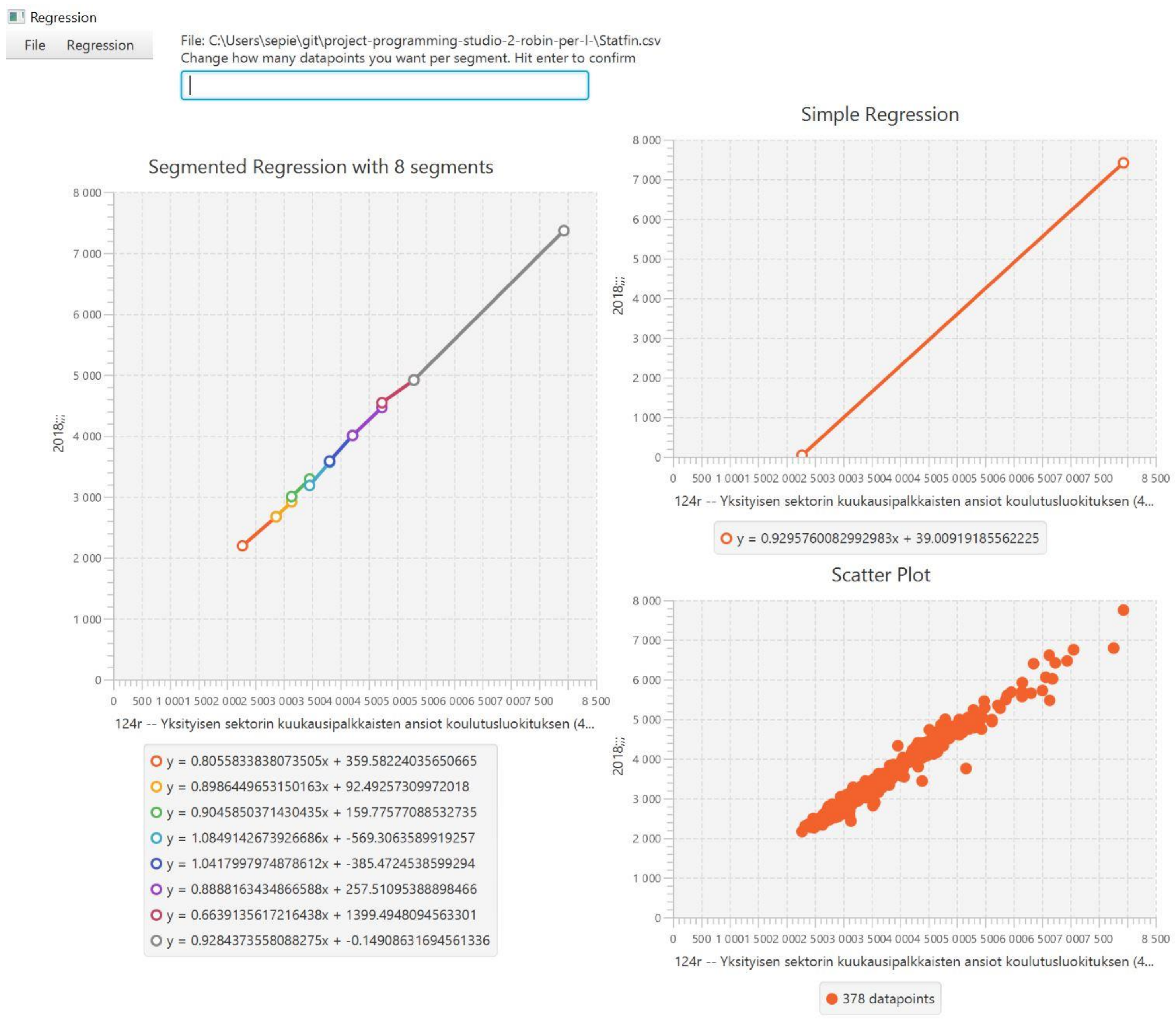
The entire program code should not be copied into the PDF document. The code is welcome to be included as an Eclipse project.



```
Current file: data3.csv
Some messages will appear here. Don't restart your program if you want your changes to stay
```

```
y = 7.363636363636365x + -52.363636363636374
```

```
y = 1.0x + 0.0
y = -2.4999999999999956x + 10.999999999999982
y = -49.000000000000206x + 256.66666666666794
y = 24.999999999999936x + -316.6666666666662
y = 74.99999999999943x + -733.3333333333277
y = 100.0x + -1000.0
```



File: C:\Users\sepie\git\project-programming-studio-2-robin-per-l-\Statfin.csv
Change how many datapoints you want per segment. Hit enter to confirm

```
y = 0.8055833838073505x + 359.58224035650665
y = 0.8986449653150163x + 92.49257309972018
y = 0.9045850371430435x + 159.77577088532735
y = 1.0849142673926686x + -569.3063589919257
y = 1.0417997974878612x + -385.4724538599294
y = 0.8888163434866588x + 257.51095388898466
y = 0.6639135617216438x + 1399.4948094563301
y = 0.9284373558088275x + -0.14908631694561336
```

```
y = 0.9295760082992983x + 39.00919185562225
```

```
● 378 datapoints
```